Lecture Notes

On

Digital Electronics & Microprocessor

Handwritten

Prepared By

Sri Shailesh kumar Nayak

Lecturer in Electronics

Number System

- A number system is simply a way to count.
- The most commonly used number systems are :-

(i) Decimal number system
(ii) Binary number System
(iii) Octal number system
(iv) Hexadecimal number system.

Base/Radix

The base/radix of a number system is defined as the number of different digits

- A number system with base or radix r will have r number of different digits from $0 \rightarrow (r-1)$.
- The number system is represented by N_b. where N - Number b - base or radix.

Decimal Number System

- This system has base '10'.
- It has 10 distinct symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

ex - $(498)_{10}$ → Here 4 is the most significant digit (MSD) & 8 is the least Significant digit (LSD)

Binary Number System

- It has base 2.
- It has two base numbers 0 and 1. These, base numbers are called Bits.
- In binary number system, group of 4 bits is known as Nibble & group of Eight bits is known as Byte.

4 bits = 1 Nibble, 8 bits = 1 Byte

- ex = 1011 (1011: MSB ↙, ↘ LSB)

MSB - Most Significant Bit
LSB - Least Significant Bit

Octal Number system

- It has a base of 8.
- It posses 8 distinct symbols (0, 1, 2, 3, 4, 5, 6, 7)

ex - $(274)_8$

Hexadecimal Number System

- The base for this system is 16
- This number system contains numeric digits (0, 1, 2, ···· 9) & alphabets (A, B, C, D, E & F) both. So this is an Alphanumeric number system.
- Microprocessor deals with instructions & data that use hexadecimal number system for programming purpose.
- ex - $(A7)_{16}$, $(9E3)_H$

Conversion

Decimal to Binary

ex convert $(57)_{10}$ to Binary equivalent.

2 | 57		
2 | 28	—	1
2 | 14	—	0
2 | 7	—	0
2 | 3	—	1
1	—	1

$(111001)_2$

Binary to Decimal

ex convert $(10110)_2$ to Decimal equivalent.

$$10110 = 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$$
$$= 1\times16 + 0\times8 + 1\times4 + 1\times2 + 0\times1$$
$$= 16 + 0 + 4 + 2 + 0$$
$$= 22$$

ex Convert $(13.125)_{10}$ to its binary equivalent.

$$\begin{array}{r|l} 2 & 13 \\ 2 & 6 \quad -1 \\ 2 & 3 \quad -0 \\ & 1 \quad -1 \end{array}$$

(1101)

$$\begin{array}{r} .125 \\ \times\ 2 \\ \hline 0.250 \\ \times\ 2 \\ \hline 0.50 \\ \times\ 2 \\ \hline 1.0 \end{array}$$

$(.001)$

$(13.125)_{10} = (1101.001)_2$

ex Convert $(10111.101)_2$ to its decimal equivalent.

$$(10111.101)_2 = 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$

$$= 1\times16 + 0 + 1\times4 + 1\times2 + 1\times1 + 1\times\frac{1}{2} + 0 + 1\times\frac{1}{8}$$

$$= 16 + 4 + 2 + 1 + 0.5 + 0.125$$

$$= 23.625$$

$(10111.101)_2 = (23.625)_{10}$

Decimal to octal

ex Convert $(259)_{10}$ to octal equivalent.

$$\begin{array}{r|l} 8 & 259 \\ 8 & 32 \quad -3 \\ & 4 \quad -0 \end{array}$$

$(403)_8$

Octal to Decimal

ex Convert $(125)_8$ to its decimal equivalent.

$$(125)_8 = 1\times8^2 + 2\times8^1 + 5\times8^0$$

$$= 1\times64 + 2\times8 + 5\times1$$

$$= 64 + 16 + 5$$

$$= 85$$

Decimal to Hexadecimal

ex Convert $(487)_{10}$ to its Hexadecimal equivalent.

16 | 487
16 | 30 — 7
1 — E ↑

$(1E7)_{16}$ or $(1E7)_H$

Hexadecimal to Decimal

ex convert $(2B6)_{16}$ to its decimal equivalent.

$$(2B6)_{16} = 2\times16^2 + B\times16^1 + 6\times16^0$$
$$= 2\times256 + 11\times16 + 6\times1$$
$$= 512 + 176 + 6$$
$$= 594$$

Binary to octal

ex Convert $(10110111)_2$ to its octal equivalent.

Note
- group 3 bit from LSB towards MSB.
- Add '0' at the MSB which is short of 3.
- write its octal equivalent.

Binary	octal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7

Ans

Add 0

$(10110111)_2 \rightarrow$ 0 | 0 1 1 | 0 1 1 1

2 6 7

$(10110111)_2 = (267)_8$

Octal to Binary

- Represent 3 bit binary equivalent of Octal number individually.

ex convert $(153)_8$ to its binary equivalent.

1 5 3

001 101 011

$(153)_8 = (001101011)_2$ or $(1101011)_2$

Binary to Hexadecimal

- Combine 4 bit from LSB to MSB & add as many 0 to MSB as short of 4.

→ write its binary equivalent.

Binary	Hexadecimal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

ex Convert $(1011101011011 0)_2$ to its Hexadecimal equivalent.

0 0 1011 1010 1101 10

2 E B 6

$(1011101011011 0)_2 = (2EB6)_{16}$ or $(2EB6)_H$

Hexadecimal to Binary

- write 4 bit binary equivalent of Hexadecimal number individually.

ex convert $(7BF)_{16}$ to its binary equivalent.

7 B F

0111 1011 1111

$(7BF)_{16} = (011110111111)_2$ or $(111101111111)_2$

Octal to Hexadecimal

- Convert octal to its binary equivalent.
- Convert binary to its Hexa decimal equivalent

Hexadecimal to octal

- convert Hexadecimal to its binary equivalent.
- Convert binary to its octal equivalent.

ex convert $(3754)_8$ to its Hexadecimal equivalent

3754

011 111 101 100

011 111 101 100

7 E C

$(3754)_8 = (7EC)_{16}$

Binary Addition

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$ or 0 with 1 carry.

ex Add $(1011)_2$ & $(1100)_2$

carry → ①

$$\begin{array}{r} 1011 \\ +\ 1100 \\ \hline 10111 \end{array}$$

ex Add $(101101)_2$ & $(001110)_2$

$$\begin{array}{r} \text{①①} \\ 101101 \\ +\ 001110 \\ \hline 111011 \end{array}$$

Binary Subtraction

$0 - 0 = 0$

$0 - 1 = 1$ (1 is borrowed & 0 becomes 10)

$1 - 0 = 1$

$1 - 1 = 0$

ex Subtract $(11)_2$ from $(100)_2$

$$\begin{array}{r} {}^{1}\!\not{1}\ {}^{10}\!\not{0}\ 0 \\ -\ 11 \\ \hline 001 \end{array}$$

ex Subtract $(1001)_2$ from $(1100)_2$

$$\begin{array}{r} {}^{1}\!\not{1}\ {}^{10}\!\not{1}\ 00 \\ -\ 1001 \\ \hline 0011 \end{array}$$

Binary Multiplication

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

ex Multiply $(110)_2$ & $(11)_2$

$$
\begin{array}{r}
110 \\
\times \quad 11 \\
\hline
110 \\
110- \\
\hline
10010
\end{array}
$$

ex Multiply $(1011)_2$ & $(101)_2$

$$
\begin{array}{r}
1011 \\
\times \quad 101 \\
\hline
1011 \\
0000- \\
1011-- \\
\hline
110111
\end{array}
$$

Binary Division

ex Divide $(11000)_2$ by $(1000)_2$

$$
\begin{array}{r|l|l}
1000 & 11000 & 11 \\
 & 1000 & \\
\hline
 & 1000 & \\
 & 1000 & \\
\hline
 & 0 &
\end{array}
$$

ex Divide $(1111000)_2$ by $(100)_2$

$$
\begin{array}{r|l|l}
100 & 1111000 & 11110 \\
 & 100 & \\
\hline
 & 111 & \\
 & 100 & \\
\hline
 & 1100 & \\
 & 100 & \\
\hline
 & 100 & \\
 & 100 & \\
\hline
 & 000 & \\
 & 000 & \\
\hline
 & 0 &
\end{array}
$$

Binary	Decimal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

1's Complement

- 1's complement of a binary number is obtained by changing 0 to 1 & 1 to 0.
- The complemented value represents the negative of the original number.

ex: Find 1's complement representation of 10110

10110
01001 ← 1's Complement

ex: Find 1's Complement of 10000

10000
01111 ← 1's Complement.

2's Complement

- Find 1's Complement, then add 1 to the 1's complement.

ex Find 2's complement of 1010.

```
          1010
1's complement → 0101
               +   1
2's complement → 0110
```

ex Find 2's complement of 101101.

```
            101101
I's complement → 010010
                 +     1
2's complement → 010011
```

Note

- A negative number can be converted into a positive number by finding its 2's complement.
- The MSB on the left most bit indicates the sign. If it is '1' the number is negative & if '0' the number is positive.

ex Represent −6 in 2's complement form.

```
Binary equivalent of 6 → 00000110
    1's complement     → 11111001
                           +   1
                       ----------
                         11111010
```

ex Represent −45 in 2's complement form

```
Binary equivalent of 6 → 00101101
    1's complement     → 11010010
                           +    1
                       ----------
                         11010011
```

Subtraction in 2's complement method

- Ignore if canny occur.
- If MSB of result is 1 then it is a negative number & If MSB is 0 then it is a positive number.
- 2's complement of result gives the original number if result is negative.

ex: Subtract 8 from 9 using 2's complement in 8-bit.

$9 - 8 = ?$

9 → 00001001

Binary of 8 → 00001000

1's complement → 11110111 (carries: 1 1 1)

\+ 1

2's complement → 11111000

Binary of 9 → 00001001 (carries: 1 1 1 1)

2's complement of 8 → + 11111000

(1) 00000001

(1) → Canny (ignore)

MSB → 0 (+ve number)

result → $(00000001)_2$ i.e $(1)_{10}$

ex: Subtract 18 from 13 using 2's complement

$13 - 18 = 2$

Binary of 18 → 00010010

1's Complement → 11101101

\+ 1

2's complement → 11101110

Binary of 13 → 00001101 (carries: 1 1)

2's complement of 18 → + 11101110

(1) 11111011

(1) → MSB → 1 i.e Negative number

result → 11111011

1's complement → 00000100

$$\begin{array}{r} + \quad\quad 1 \\ \hline \end{array}$$

2's complement → 00000101

$(00000101)_2 = (5)_{10}$

i.e result is '−5'.

ex: Add −15 & −20

Binary of 15 → 00001111

1's complement → 11110000

$$\begin{array}{r} + \quad\quad 1 \\ \hline \end{array}$$

2's complement → 11110001

Binary of 20 → 00010100

1's complement → 11101011

$$\begin{array}{r} + \quad\quad 1 \\ \hline \end{array}$$

2's complement → 11101100

2's complement of 15 i.e −15 → 11110001

" " " 20 i.e −20 → + 11101100

(1)11011101

Carry (Ignore)

MSB → 1 i.e Negative

result → 11011101

1's complement → 00100010

$$\begin{array}{r} + \quad\quad 1 \\ \hline \end{array}$$

2's complement → 00100011

$(00100011)_2 = (35)_{10}$

i.e result is '−35'

Weighted Binary Codes

In weighted codes, for each position, there is specific weight attached.

Binary Coded Decimal CBCD

- In this code, each digit of a decimal number is represented by binary equivalent.
- It is a 4-bit binary code.
- It is also known as '8-4-2-1 code' or simply 'BCD code'.
- It is a weighted code system.
- ex Express $(943)_{10}$ in BCD or 8421 code.

9 4 3
↓ ↓ ↓
1001 0100 0011

$(100101000011)_{BCD}$

BCD (8421)	Binary	Decimal
0000	0000	0
0001	0001	1
0010	0010	2
0011	0011	3
0100	0100	4
0101	0101	5
0110	0110	6
0111	0111	7
1000	1000	8
1001	1001	9
0001 0000	1010	10
0001 0001	1011	11
0001 0010	1100	12
0001 0011	1101	13
0001 0100	1110	14
0001 0101	1111	15

ex: Convert $(00110010.10010100)_{BCD}$ to its decimal equivalent

0011 0010 . 1001 0100
↓ ↓ ↓ ↓
3 2 9 4

$(32.94)_{10}$

ex: Express $(214.83)_{10}$ to its BCD equivalent.

214 . 83
↙ ↓ ↘ ↘ ↘
0010 0001 0100 1000 0011

Non-weighted codes

In non-weighted code, there is no positional weighted. i-e each position within the binary number is not assigned a prefixed value.

Excess-3 code

In excess-3 (XS-3) code three (3) is added to each decimal digit before converting it into equivalent binary.

- Each 4 bit group in excess-3 code is equal to a specific decimal digit.

Decimal	Excess-3 code	Binary
0	0011	0000
1	0100	0001
2	0101	0010
3	0110	0011
4	0111	0100
5	1000	0101
6	1001	0110
7	1010	0111
8	1011	1000
9	1100	1001
10	0100 0011	1010
11	0100 0100	1011
12	0100 0101	1100
13	0100 0110	1101
14	0100 0111	1110
15	0100 1000	1111

ex: Express 129 to an excess-3 number.

1	2	9
+3	+3	+3
4	5	12
↓	↓	↓
0100	0101	1100

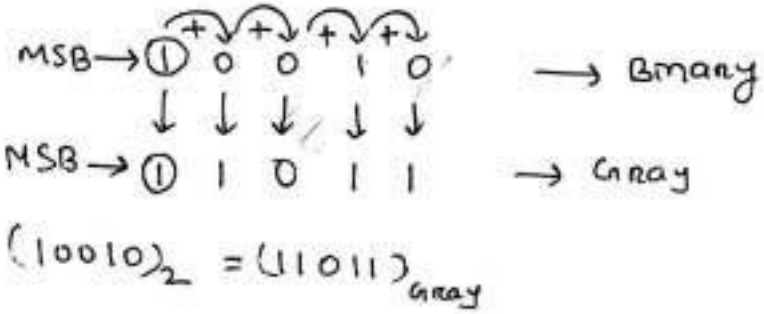
Gray Codes

Decimal	Binary	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary to Gray Conversion

- MSB in the gray code is same as corresponding digit in binary number.
- Starting from 'left to Right', add each adjacent pair of binary bits to get next gray code bit, discarding the carry.

ex Convert $(10010)_2$ to gray code.

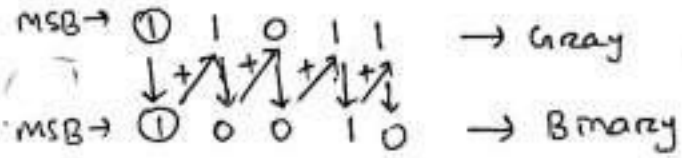


$(10010)_2 = (11011)_{Gray}$

Gray to Binary Conversion

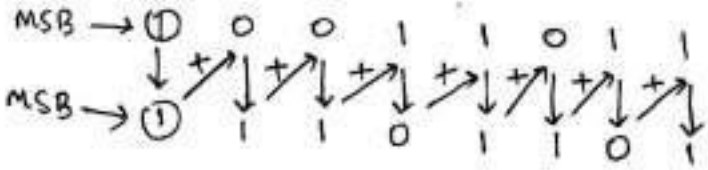
- MSB of Binary is same as that of gray code.
- Add each binary bit to the gray code bit of the next adjacent position to get next bit of the binary number, discarding the carry.

ex Convert $(11011)_{Gray}$ to Binary code.



$(11011)_{Gray} = (10010)_2$

ex Convert $(1001\ 1011)_{Gray}$ to Binary code.



$(10011011)_{Gray} = (11101101)_2$

Error Detecting Code

- In digital systems a word consisting of group of bits is stored as a unit & moves from one unit to another.
- When this word is transmitted from one memory location to another or to an arithmetic unit, an error may occur.

ex: If a task of performing addition of 01001101 & 11000111 is required & when the device transfer these words from memory, it may possible that error in 3rd bit of first word can occure as 01101101. This will result in an error in the addition.

- For detecting such errors, a method called 'Parity' is used.

Parity

- For detecting such error, an additional bit known as parity bit is added with the number.
- For odd parity, this parity bit is set to 1 so that the sum of bits in the number is odd.
- For even parity, the adding of the parity bit to the group of bits produce an even number of 1's.

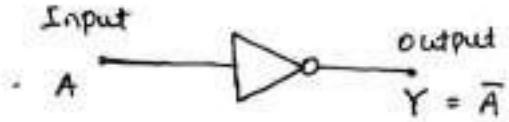
Decimal	BCD code	Even parity	Odd parity
0	0000	0	1
1	0001	1	0
2	0010	1	0
3	0011	0	1
4	0100	1	0
5	0101	0	1
6	0110	0	1
7	0111	1	0
8	1000	1	0
9	1001	0	1

Logic Gates

Basic Gates	Universal Gate	Special purpose Gate
NOT	NAND	EX-OR (XOR)
AND	NOR	EX-NOR (XNOR)
OR		

NOT Gate

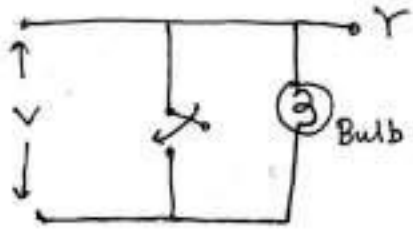
- The NOT gate has a single input variable and a single output variable.
- The NOT operation is also referred to as 'INVERSION' or 'COMPLEMENTATION'.
- Thus its output logic level is always opposite to the logic level of its input.



(Symbol)

Input	output
A	Y
0	1
1	0

(Truth Table)

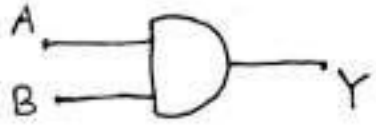


(Switching circuit)

- For input A, $\boxed{Y = \bar{A}}$

AND Gate

- The AND gate can have two or more inputs but only one output.
- If all the inputs or any of the input is '0', the output is '0'. The output is '1' only when all the inputs are '1'.

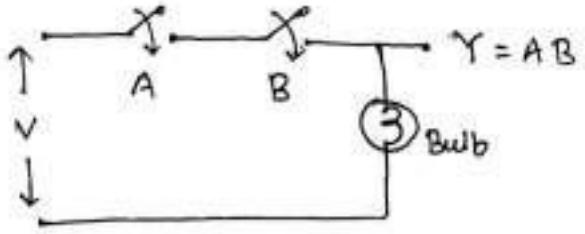


(Logic Symbol)

Input		output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(Truth Table)

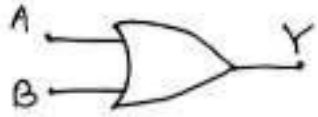
- The logical expression is $\boxed{Y = A \cdot B}$



(Switching circuit)

OR Gate

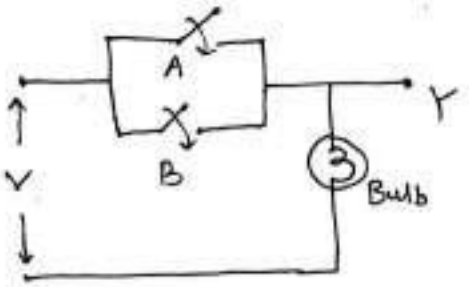
- The OR gate can have two or more inputs but only one output
- If all the inputs or any of the input is '1' the output it high OR '1'. If all the inputs are low or '0' then output is '0'.



(Logic Symbol)

Input		output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(Truth Table)

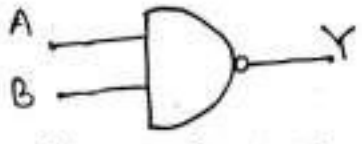


(Switching circuit)

- The logical expression is $\boxed{Y = A + B}$

NAND Gate

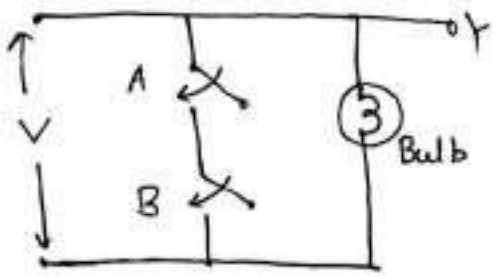
- It may have two or more inputs but only one output.
- The NAND Gate is a AND Gate followed by NOT Gate. It is a NOT-AND operation
- The output is 1 when either one of the input or when both the inputs are at logic '0'
- The NAND gate output is exactly inverse of the AND gate.



(Logic Symbol)

(Truth Table)

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



(Switching circuit)

- The logical expression is $\boxed{Y = \overline{A \cdot B}}$

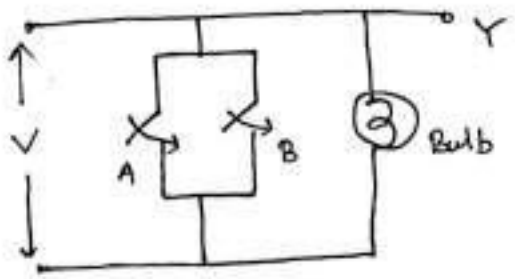
NOR Gate

- It may have two or more input and an output.
- A NOR gate is a combination of OR gate & NOT Gate. It is a NOT-OR operation.
- The output is '1' only if all the inputs are at logic '0'.

(Logic symbol)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

(Truth Table)



(Switching Circuit)

- The logical expression is $Y = \overline{A+B}$

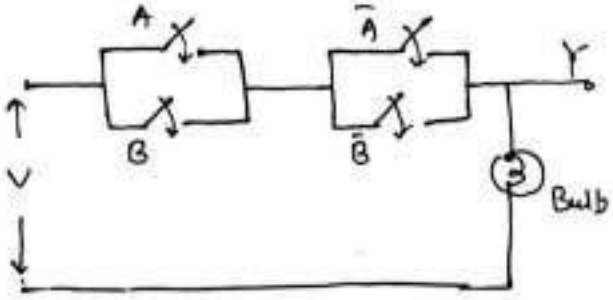
EX-OR Gate

- It is a two input single output logic gate.
- output is high or '1' only when only one of its inputs is high (1).
- It is also known as 'stair case switch'.

(Logic Symbol)

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(Truth Table)



(Switching circuit)

- Logical Expression is $\boxed{Y = A \oplus B}$
- $A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B$

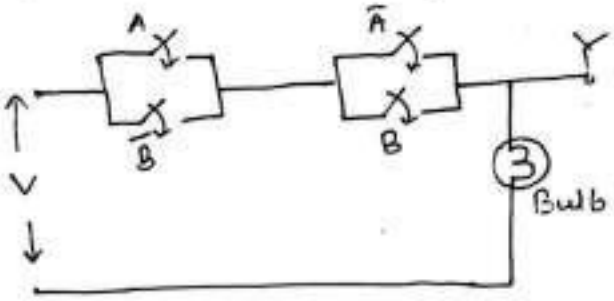
EXNOR Gate

- It is a two input and one output logic circuit.
- output is '1' only when both the inputs are same.
- It is also called gate of equivalence or 'coincidence logic'.

(logic symbol)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

(Truth Table)



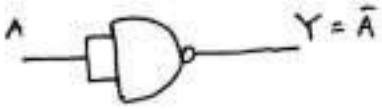
(Switching circuit)

- logical Expression is $\boxed{Y = A \odot B}$
- $A \odot B = A.B + \bar{A}.\bar{B}$

Realization of logic gates using universal gate (NAND, NOR)

NOT gate realization

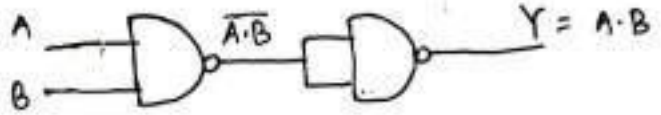
Using NAND :



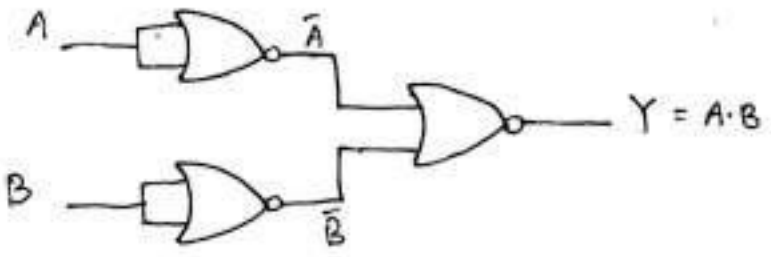
Using NOR :

AND gate realization

Using NAND :-

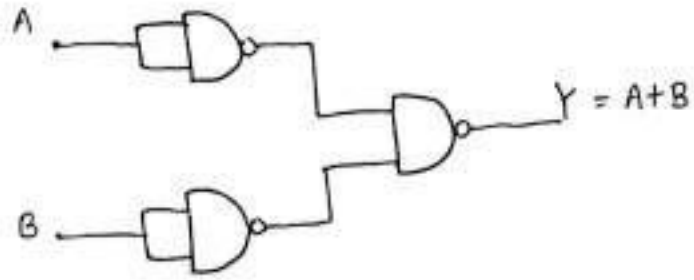


Using NOR :-

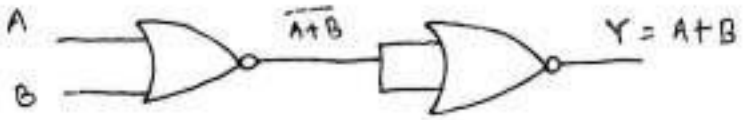


OR gate realization

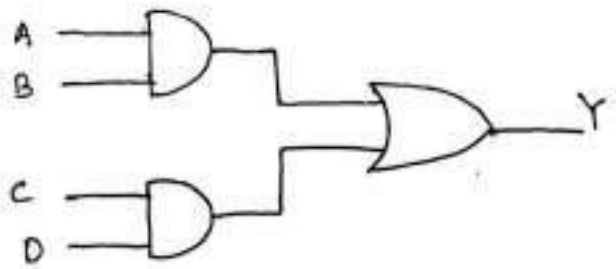
Using NAND :-



Using NOR :-



Ex Realize AB + CD using logic gates.



Boolean Algebra

- The binary operations performed by any digital circuit with the set of elements 0 & 1, are called logical operations or logic functions.
- The algebra used to symbolically represent the logic function is called Boolean algebra.
- Boolean algebra is a system of mathematic logic for the analysis & designing of digital system.

AND operation

$$A \cdot A = A$$
$$A \cdot 0 = 0$$
$$A \cdot 1 = A$$
$$A \cdot \bar{A} = 0$$

OR Operation

$$A + A = A$$
$$A + 0 = A$$
$$A + 1 = 1$$
$$A + \bar{A} = 1$$

NOT Operation

$$\bar{\bar{A}} = A \qquad \boxed{\bar{1} = 0} \qquad \boxed{\bar{0} = 1}$$

Laws of Boolean Algebra

Commutative Laws

(i) $A + B = B + A$

(ii) $A \cdot B = B \cdot A$

Associative Laws

(i) $(A + B) + C = A + (B + C)$

(ii) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

Distributive Laws

(i) $A(B+C) = AB + AC$

(ii) $A + BC = (A+B)(A+C)$

Idempotence Law

(i) $A \cdot A = A$

(ii) $A + A = A$

Absorption Law

(i) $A + AB = A(1+B) = A$

(ii) $A(A+B) = A$

Involutionary Law

$\bar{\bar{A}} = (A')' = A$

De Morgan's Theorem

(i) The complement of the product of variables is equal to the sum of their individual complements.

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

(ii) The complement of a sum of variables is equal to the product of their individual complements.

$$\overline{A+B} = \bar{A} \cdot \bar{B}$$

Transposition Theorem

$$AB + \bar{A}C = (A+C)(\bar{A}+B)$$

Proof RHS $= (A+C)(\bar{A}+B)$

$= A\bar{A} + \bar{A}C + AB + BC$

$= 0 + \bar{A}C + AB + BC(A+\bar{A})$

$= AB + ABC + \bar{A}C + \bar{A}CB$

$= AB(1+C) + \bar{A}C(1+B)$

$= AB + \bar{A}C = LHS$

Proof of Demorgan's law

A	B	$\bar{A}$	$\bar{B}$	$A \cdot B$	$\overline{A \cdot B}$	$\bar{A} + \bar{B}$	$A + B$	$\overline{A + B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1	0	1	1	0	1	1
0	1	1	0	0	1	1	1	0	0
1	0	0	1	0	1	1	1	0	0
1	1	0	0	1	0	0	1	0	0

Simplification of logic expression using Boolean algebra :

ex Solve the following

(i) $X \cdot X \cdot Y + X \cdot Y \cdot Y + Y \cdot \bar{Y} \cdot X \cdot X$

(ii) $A \cdot \bar{B} + \bar{A} \cdot B + A \cdot B + \bar{A} \cdot \bar{B}$

(iii) $(\bar{A} + \bar{B}) \cdot A\bar{B}C$

(iv) $P + \bar{P}Q\bar{R} + \overline{\bar{Q} + R}$

(v) $\overline{X\bar{Y} + XYZ} + X(Y + X \cdot \bar{Y})$

Ans

(i) $X \cdot X \cdot Y + X \cdot Y \cdot Y + Y \cdot \bar{Y} \cdot X \cdot X$

$= XY + XY + 0 \qquad (X \cdot X = X,\ Y \cdot Y = Y,\ Y \cdot \bar{Y} = 0)$

$= XY \qquad (XY + XY = XY)$

(ii) $A \cdot \bar{B} + \bar{A} \cdot B + A \cdot B + \bar{A} \cdot \bar{B}$

$= A(\bar{B} + B) + \bar{A}(B + \bar{B})$

$= A \cdot 1 + \bar{A} \cdot 1 \qquad (B + \bar{B} = 1)$

$= A + \bar{A} \qquad (A \cdot 1 = A)$

$= 1$

(iii) $(\bar{A} + \bar{B})\, A \cdot \bar{B} \cdot C$

$= \bar{A} \cdot A \cdot \bar{B} \cdot C + \bar{B} \cdot A \cdot \bar{B} \cdot C$

$= 0 + A\bar{B} \cdot C \qquad (\bar{A} \cdot A = 0,\ \bar{B} \cdot \bar{B} = B)$

$= A\bar{B}C$

(iv) $P + \bar{P}Q\bar{R} + \overline{Q+R}$

$= P + \bar{P}Q\bar{R} + \bar{Q}\cdot\bar{R}$

$= P + \bar{R}(\bar{P}Q + \bar{Q})$

$= P + \bar{R}(\bar{P}+\bar{Q})(Q+\bar{Q})$

$= P + \bar{R}(\bar{P}+\bar{Q})\cdot 1 \qquad (Q+\bar{Q}=1)$

$= P + \bar{R}(\bar{P}+\bar{Q})$

$= P + \bar{P}\bar{R} + \bar{R}\bar{Q}$

$= (P+\bar{P})(P+\bar{R}) + \bar{R}\bar{Q}$

$= P + \bar{R} + \bar{R}\bar{Q} \qquad (P+\bar{P}=1)$

$= P + \bar{R}(1+\bar{Q})$

$= P + \bar{R}$

(v) $\overline{\overline{X\bar{Y} + XYZ + X(Y + X\cdot\bar{Y})}}$

$= \overline{\overline{X(\bar{Y}+YZ) + X(Y+X)(Y+\bar{Y})}}$

$= \overline{\overline{X(\bar{Y}+Y)(\bar{Y}+Z) + X(Y+X)}} \qquad (Y+\bar{Y}=1)$

$= \overline{\overline{X(\bar{Y}+Z)} + X\cdot Y + X\cdot X}$

$= \overline{\overline{X\bar{Y} + XZ} + XY + X}$

$= \overline{\overline{X\cdot\bar{Y}\cdot\overline{XZ}} + X(Y+1)} \qquad (\overline{A+B} = \bar{A}\cdot\bar{B} \Rightarrow \text{De Morgan's law})$

$= \overline{(\bar{X}+\bar{\bar{Y}})(\bar{X}+\bar{Z}) + X}$

$= \overline{\bar{X}\cdot\bar{X} + \bar{X}\cdot\bar{Z} + \bar{X}\cdot\bar{\bar{Y}} + \bar{Z}\bar{\bar{Y}} + X}$

$= \overline{\bar{X} + \bar{X}\cdot\bar{Z} + \bar{X}\cdot Y + \bar{Z}Y + X}$

$= \overline{\bar{X}(1+\bar{Z}+Y) + X + \bar{Z}Y}$

$= \overline{\bar{X} + X + \bar{Z}Y}$

$= \overline{1 + \bar{Z}Y} \qquad (X+\bar{X}=1)$

$= \overline{1} \qquad (1+A=1 \Rightarrow \text{OR operation})$

$= 0$

(vi)

Canonical form

All the terms contain all the variables either in complementary or in uncomplementary form.

ex $f(A,B,C) = \bar{A}BC + ABC + \bar{A}B\bar{C}$

Minterm

Minterm is a product term, it contains all the variables either complementary or uncomplementary form for that combination the function output must be '1'.

- In minterms we assign '1' to each uncomplemented variable & '0' to each complemented variable.

Maxterm

Maxterm is a sum term, it contains all the variables either complementary or uncomplementary form for that combination the function output must be 0.

- In maxterms we assign '0' to each uncomplemented variable & 1 to each complemented variable.

A	B	C	Minterm	Maxterm
0	0	0	$m_0 = \bar{A}\bar{B}\bar{C}$	$M_0 = A+B+C$
0	0	1	$m_1 = \bar{A}\bar{B}C$	$M_1 = A+B+\bar{C}$
0	1	0	$m_2 = \bar{A}B\bar{C}$	$M_2 = A+\bar{B}+C$
0	1	1	$m_3 = \bar{A}BC$	$M_3 = A+\bar{B}+\bar{C}$
1	0	0	$m_4 = A\bar{B}\bar{C}$	$M_4 = \bar{A}+B+C$
1	0	1	$m_5 = A\bar{B}C$	$M_5 = \bar{A}+B+\bar{C}$
1	1	0	$m_6 = AB\bar{C}$	$M_6 = \bar{A}+\bar{B}+C$
1	1	1	$m_7 = ABC$	$M_7 = \bar{A}+\bar{B}+\bar{C}$

Sum of Product form (SOP)

- The SOP expression usually takes the form of two or more variables ANDed together.
- SOP forms are used to write logical expression for the output becoming logic '1'.

Notation :- $f(A,B,C) = \Sigma m(3,5,6,7)$

$$Y = m_3 + m_5 + m_6 + m_7$$

$$Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Product of sum form (POS)

- The POS expression usually takes the form of two or more ORed variables within parentheses, ANDed with two or more such terms.
- POS forms are used to write logical expression for output becoming logic '0'.

Notation :-

$$f(A,B,C) = \pi M(0,1,2,4)$$

$$Y = M_0 \times M_1 \times M_2 \times M_3$$

$$Y = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)$$

Standard Sum of product form

- It is also called canonical SOP form.

ex $Y = A + B\bar{C}$. Represent in canonical form

$$Y = A + B\bar{C} = A(B+\bar{B})(C+\bar{C}) + B\bar{C}(A+\bar{A})$$

$$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + AB\bar{C} + \bar{A}B\bar{C}$$

$$= ABC + A\bar{B}C + A\bar{B}\bar{C} + AB\bar{C} + \bar{A}B\bar{C}$$

Standard product of Sum form

- It is also called canonical POS form.

ex Represent $Y = (B+\bar{C})(A+\bar{B})$ in canonical form

$$Y = (B+\bar{C}+A\bar{A})(A+\bar{B}+C\bar{C})$$

$$= (A+B+\bar{C})(\bar{A}+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C})$$

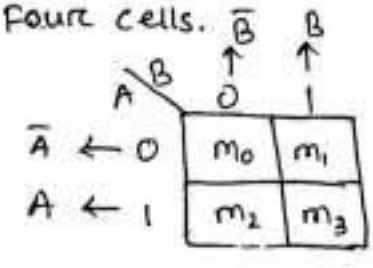
Karnaugh Map (K-Map)

The Karnaugh map is a graphical method which provides a systematic method for simplifying the Boolean expressions

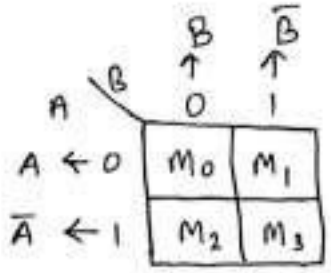
- In this technique, the information contained in a truth table or available in SOP or POS form is represented on K-map.
- In n-variable K-map there are 2^n cells.
- Gray code has been used for the identification of cells.

Two variable K-map

- Four cells.



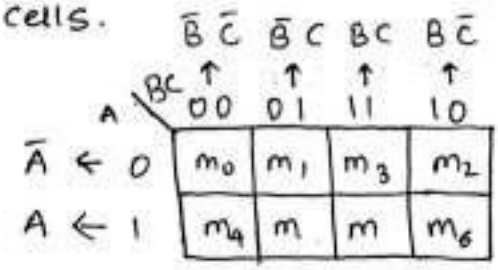
(for SOP)



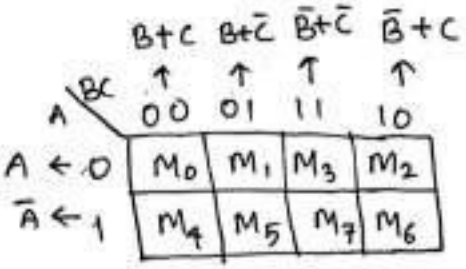
(for POS)

Three variable K-map

- Eight cells.



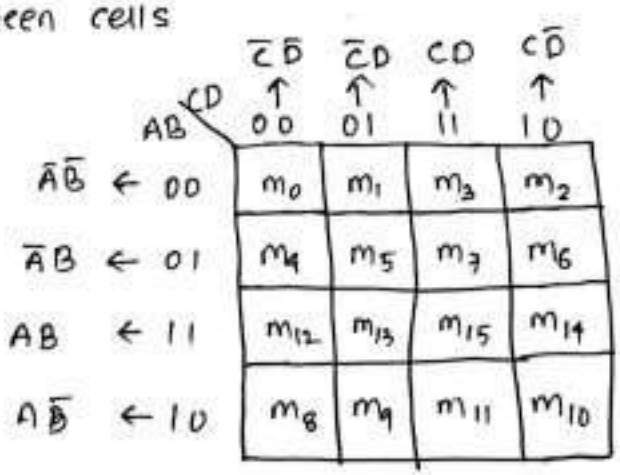
(For SOP)



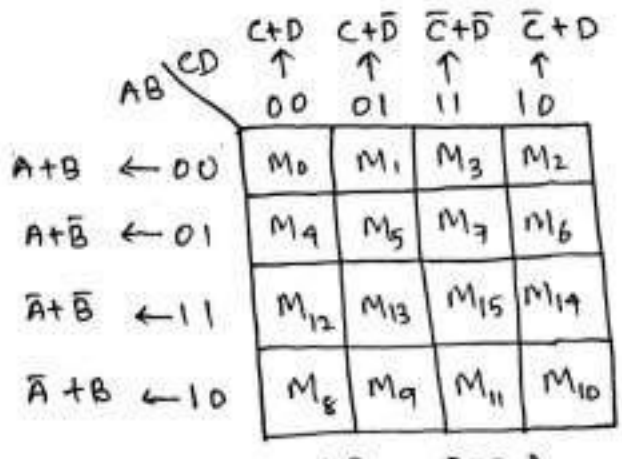
(For POS)

Four variable K-map

- Sixteen cells



(For SOP)



(For POS)

Simplification of logical functions using K-map

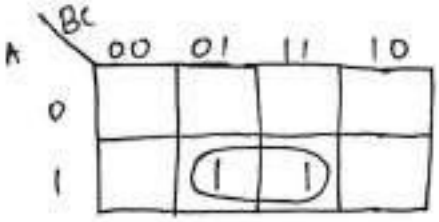
Simplification of logical functions with K-map is based on the principle of combining terms in adjacent cells.

Looping

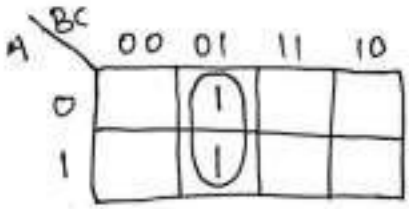
- The expression for output Y can be simplified by properly combining those cells in the K-map which contains '1's for SOP or 0's for POS. The process of combining these 1's or 0's is called looping.
- Groups are made up of 2, 4, 8, 16 & so on.
- By folding K-map over its edges, the number of 1's or 0's overlapping forms the group.

Looping groups of two (pairs)

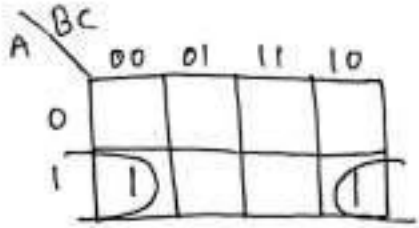
Looping a pair of adjacent 1's in a K-map eliminates the variable that appears in complemented & uncomplemented form.



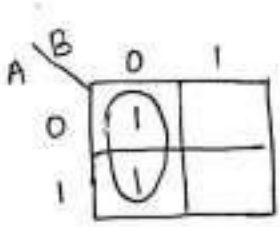
$Y = AC$



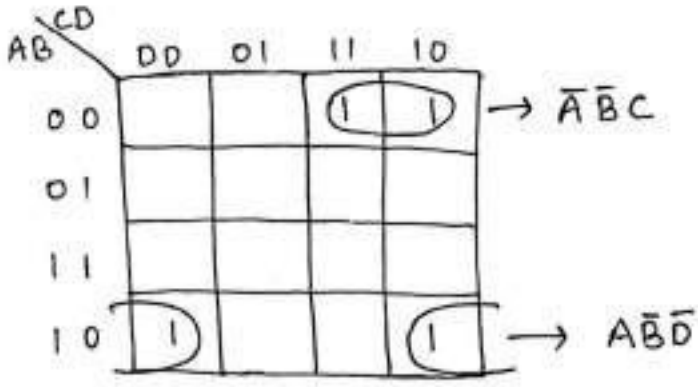
$Y = \bar{B}C$



$Y = A\bar{C}$



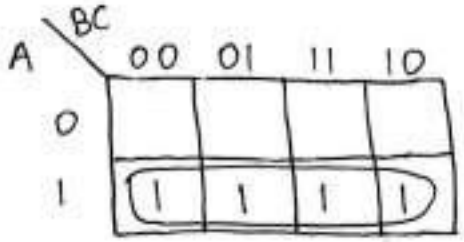
$Y = \bar{B}$



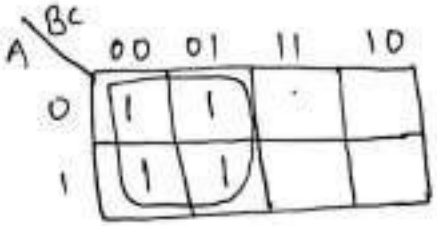
$Y = \bar{A}\bar{B}C + A\bar{B}\bar{D}$

Looping groups of four (Quads)

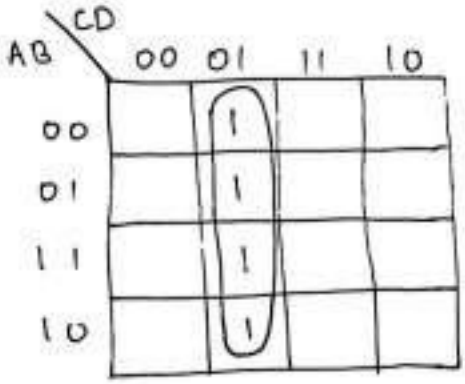
Looping a quad of 1's eliminates those two variables that appear in both complemented & uncomplemented form.



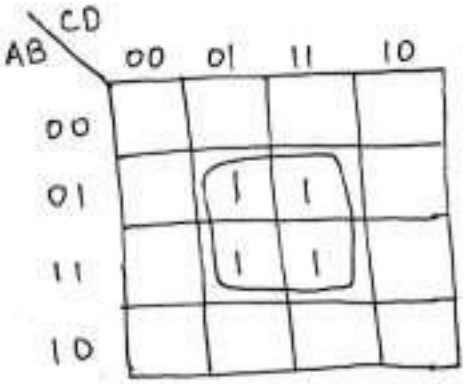
$Y = A$



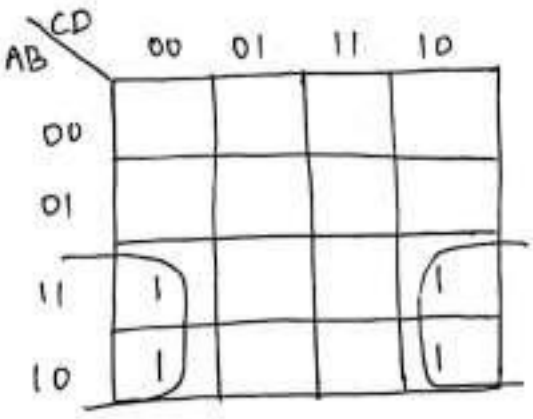
$Y = \bar{B}$



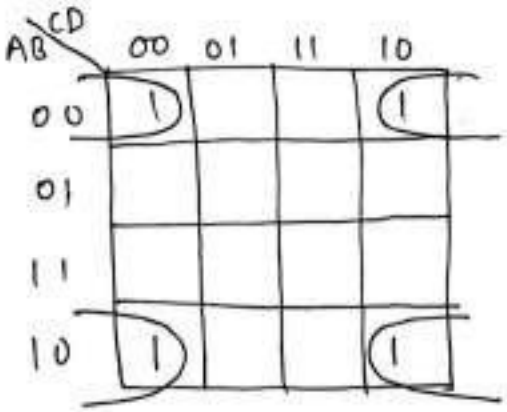
$Y = \bar{C}D$



$Y = BD$



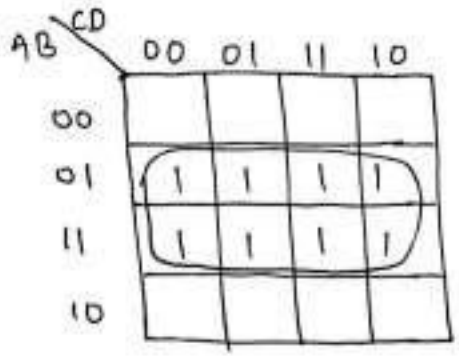
$Y = A\bar{D}$



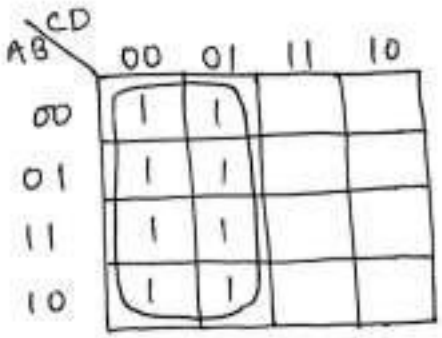
$Y = \bar{B}\bar{D}$

Looping groups of eight (octets)

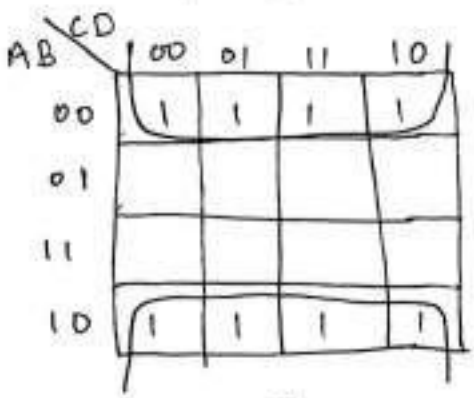
Looping an octet of 1's eliminates those three variables that appear in both complemented & uncomplemented form.



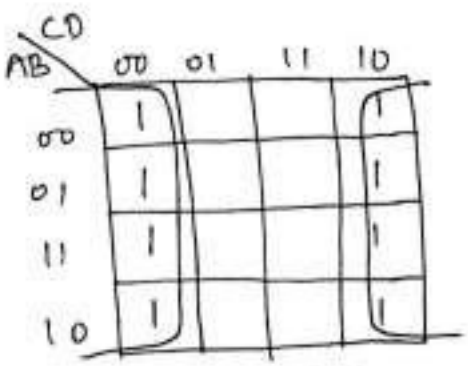
$Y = B$



$Y = \bar{C}$



$Y = \bar{B}$



$Y = \bar{D}$

Simplification rules

- Construct the K-map & place 1's in those cells corresponding to the 1's in the truth table.
- Examine the map for adjacent 1's & loop those 1's which are not adjacent to any other 1's.
- Look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
- Loop any octet even it contains some 1's that have already been looped.
- Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum no. of loops.
- Loop any pairs necessary to include any 1's that have not yet been looped, making sure to use the minimum number of loops.
- Form the OR sum of all the terms generated by each loop.

Implicant

Implicant is a product term on the given function, for that combination the function output must be 1.

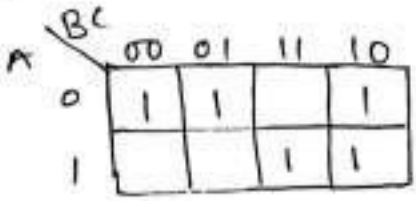
Prime Implicant (PI)

Prime implicant is a smallest possible product term on the given function, removing any one of the literal from which is not possible.

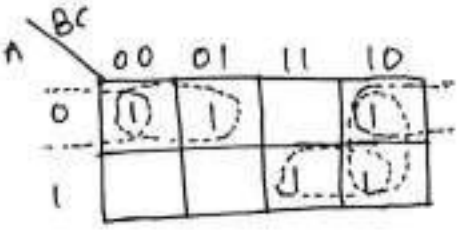
Essential Prime Implicant (EPI)

Essential prime implicant is a prime implicant, it must cover atleast one minterm, which is not covered by any other prime implicant.

ex For the given K-map. Find implicant, Prime implicant & Essential Prime implicant.



Ans



Implicant = total no. of 1 = 5

Implicant = $\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{B}C$, ABC, $\bar{A}B\bar{C}$, $AB\bar{C}$

Prime Implicant = $\bar{A}\bar{B}$, $\bar{A}\bar{C}$, AB, $B\bar{C}$

Essential Prime Implicant = $\bar{A}\bar{B}$, AB

Don't Care condition

- Some logic circuits can be designed so that there are certain input combinations for which there are no specified output levels, usually because these input combinations will never occur.
- So a circuit designer is free to make the output for any 'don't care' condition either a '0' or '1' in order to produce the simplest output expression.
- It is denoted as 'd' or 'x'.
- Mapping of don't care is not compulsory.

ex Solve the following

(i) In terms of SOP & don't care conditions.

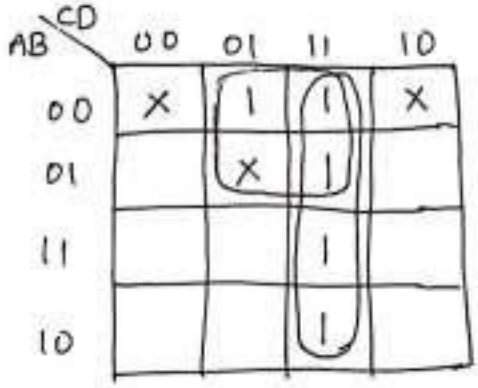
$$f(A,B,C,D) = \Sigma m(1,3,7,11,15) + d(0,2,5)$$

(ii) In terms of POS & don't care conditions

$$f(A,B,C,D) = \Pi M(4,5,6,7,8,12) \cdot d(1,2,3,9,11,14)$$

Solution

(i) $f(A,B,C,D) = \Sigma m(1,3,7,11,15) + d(0,2,5)$



$$Y = \bar{A}D + CD$$

(ii)

AB \ CD	00	01	11	10
00		d	d	d
01	0	0	0	0
11	0			d
10	0	d	d	

$Y = (A + \bar{B})(\bar{A} + C + D)$

Ex Solve the following function using K-map & realise the reduced function using (i) NAND gates (ii) NOR gates

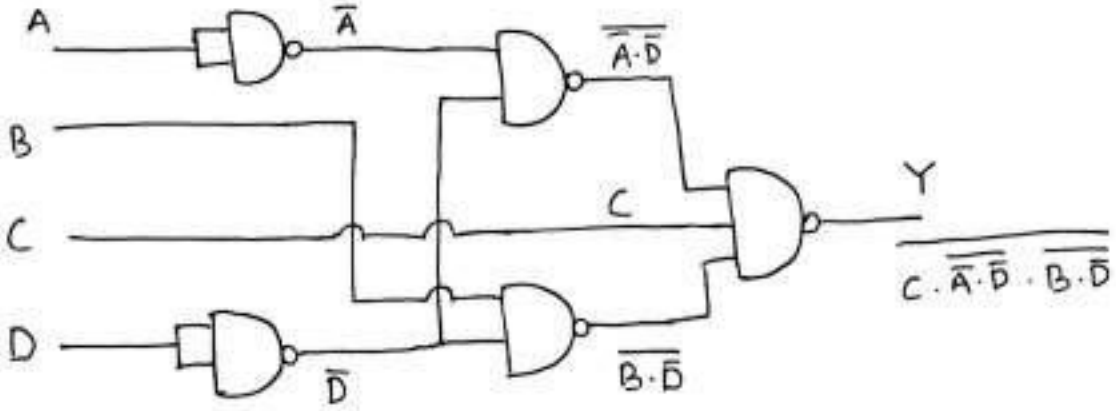
(i) $F = \Sigma m(0,1,4,5,12,13,8,9,2,6,14)$

AB \ CD	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

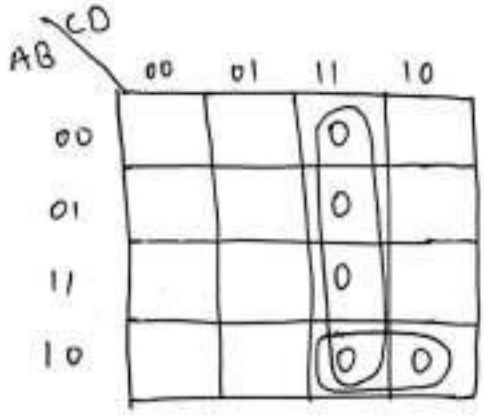
$Y = \bar{C} + \bar{A}\bar{D} + B\bar{D}$

For NAND gate realization, we use $\bar{\bar{A}} = A$

$$Y = \overline{\overline{\bar{C} + \bar{A}\bar{D} + B\bar{D}}}$$
$$= \overline{\bar{\bar{C}} \cdot \overline{\bar{A}\bar{D}} \cdot \overline{B\bar{D}}}$$
$$= \overline{C \cdot \overline{\bar{A}\bar{D}} \cdot \overline{B\bar{D}}}$$



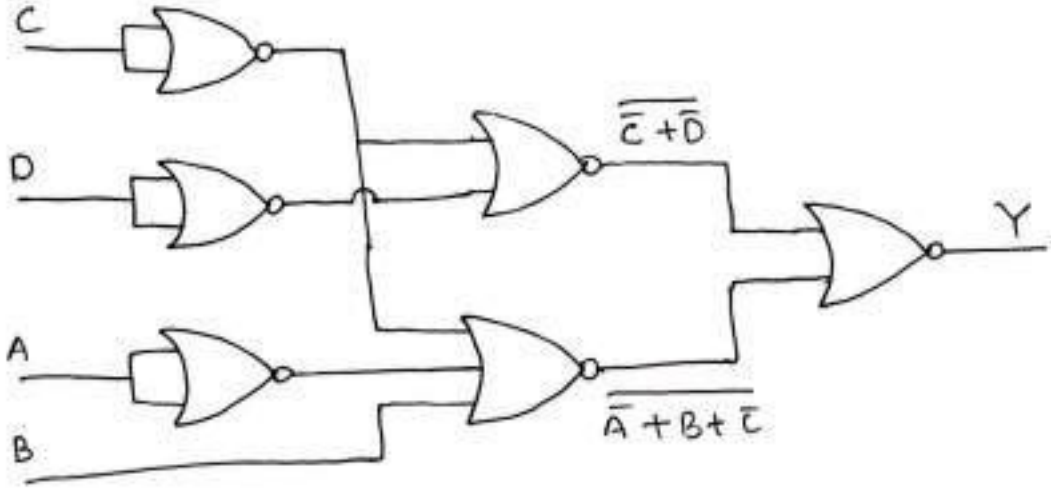
(ii)



$$Y = (\bar{C} + \bar{D}).(\bar{A} + B + \bar{C})$$

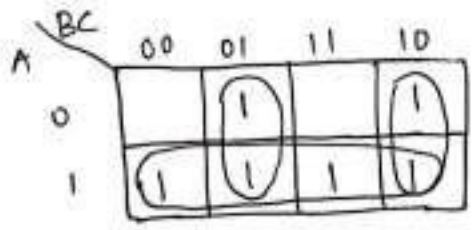
$$= \overline{\overline{(\bar{C} + \bar{D}).(\bar{A} + B + \bar{C})}}$$

$$= \overline{\overline{(\bar{C} + \bar{D})} + \overline{(\bar{A} + B + \bar{C})}}$$



ex Simplify the Boolean function, $F = \Sigma m(1, 2, 4, 5, 6, 7)$.

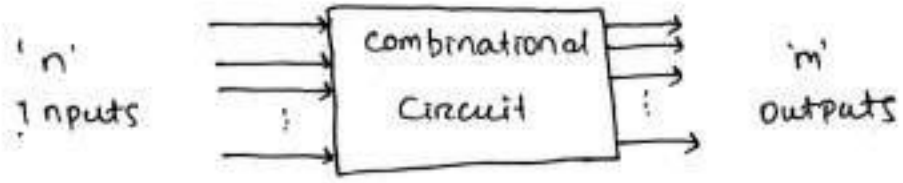
Soln



$$Y = A + \bar{B}C + B\bar{C}$$

Combinational Logic Circuit

- A combinational circuit consists of an interconnection of logic gates, whose outputs at any instant of time are determined from present combination of inputs only.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- The combinational circuit can have an 'n' number of inputs and 'm' number of outputs.

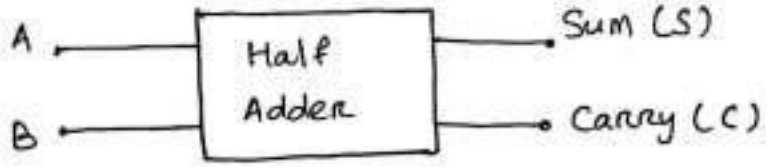


combinational circuit design

(i) Identify number of inputs & outputs.
(ii) Construct truth table.
(iii) Write output logical expression.
(iv) Minimize logical expression.
(v) Implement logic circuit.

Half Adder

- The half adder is an arithmetic circuit used to perform the addition of two single bits.
- Two input variables to the half adder designate the augend & addend bits and the output variables produce the sum & carry.



A, B → Input

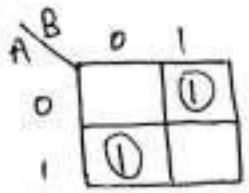
S, C → Output

Truth Table

Inputs		outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

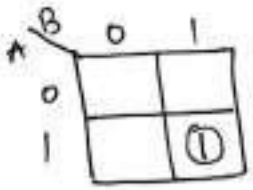
The logical expression for S & C is obtaind using k-map.

Sum (S)



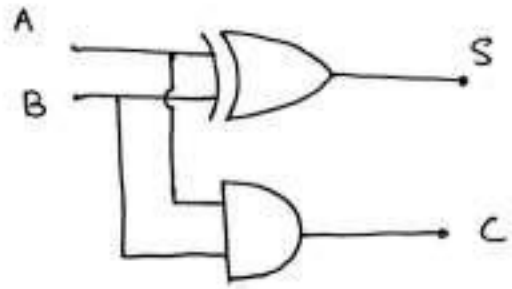
$S = \bar{A}B + A\bar{B} = A \oplus B$

Carry (C)

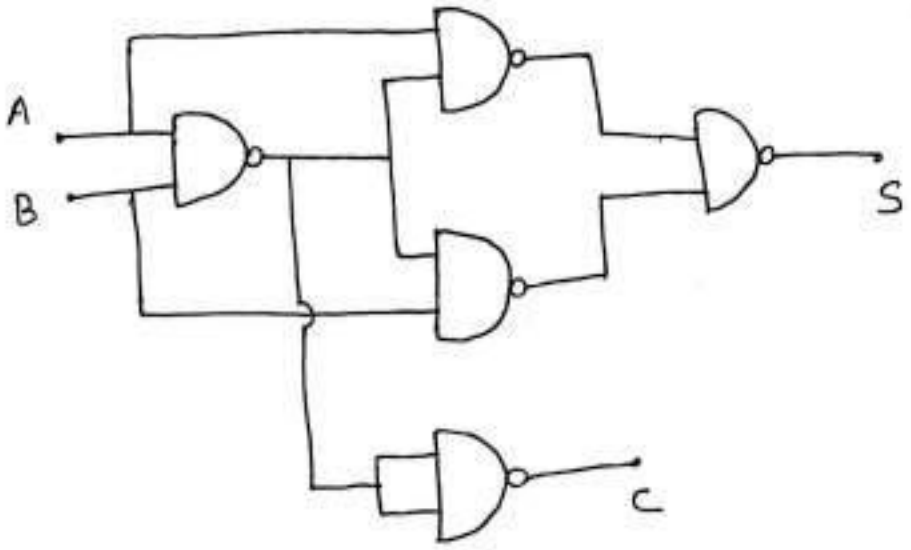


$C = AB$

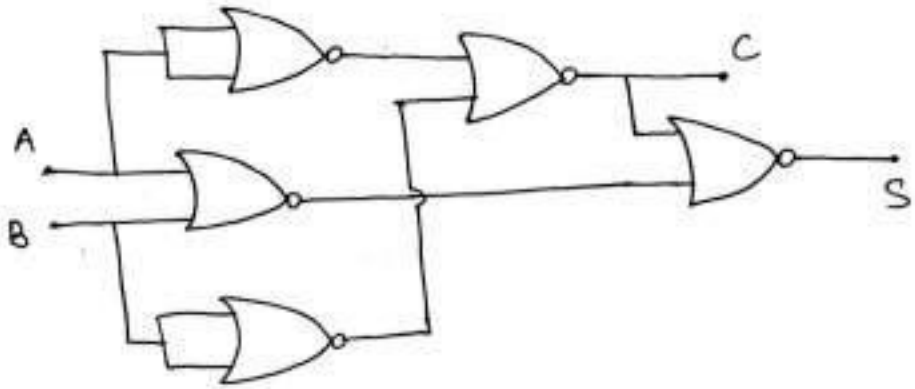
logic diagram



Half Adder using NAND gates only :-

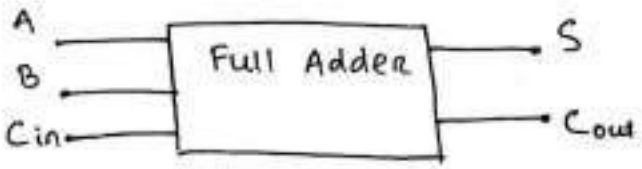


Half Adder using NOR gates only



Full Adder

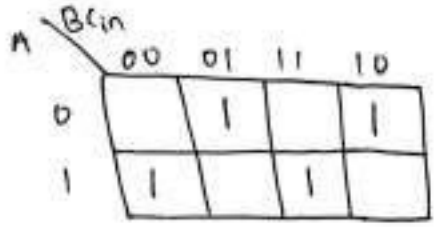
- A Full adder is a combinational circuit that performs the arithmetic sum of three input bits. It consists of three input variables designated by augend, addend & the carry bit. The two output variables produce the sum & carry.
- The third input C represents the carry from the previous lower significant position.



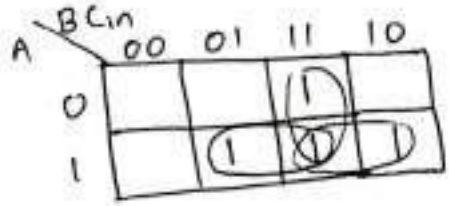
Truth Table

Inputs			output	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum (S)



Carry (Cout)



$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

$= \bar{A}(\bar{B}C_{in} + B\bar{C}_{in}) + A(\bar{B}\bar{C}_{in} + BC_{in})$

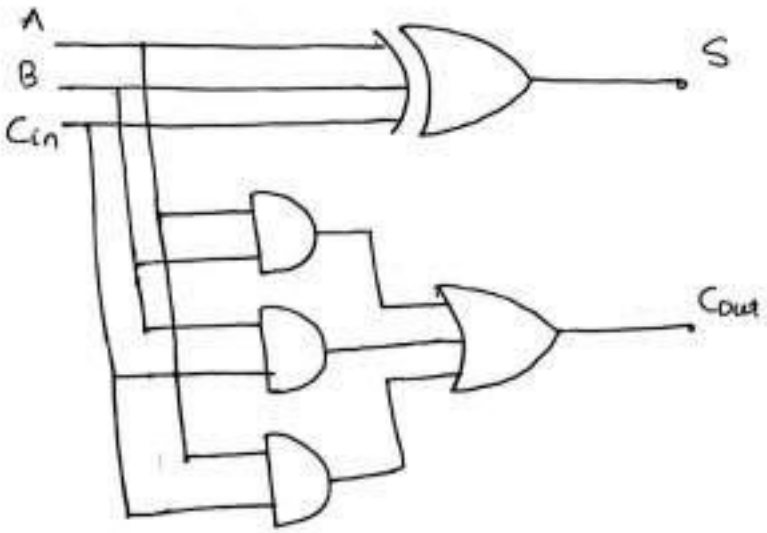
$= \bar{A}(B \oplus C_{in}) + A(B \odot C_{in})$

$= \bar{A}(B \oplus C_{in}) + A(\overline{B \oplus C_{in}})$

$= A \oplus B \oplus C_{in}$

$C_{out} = AC_{in} + AB + BC_{in}$

logic diagram



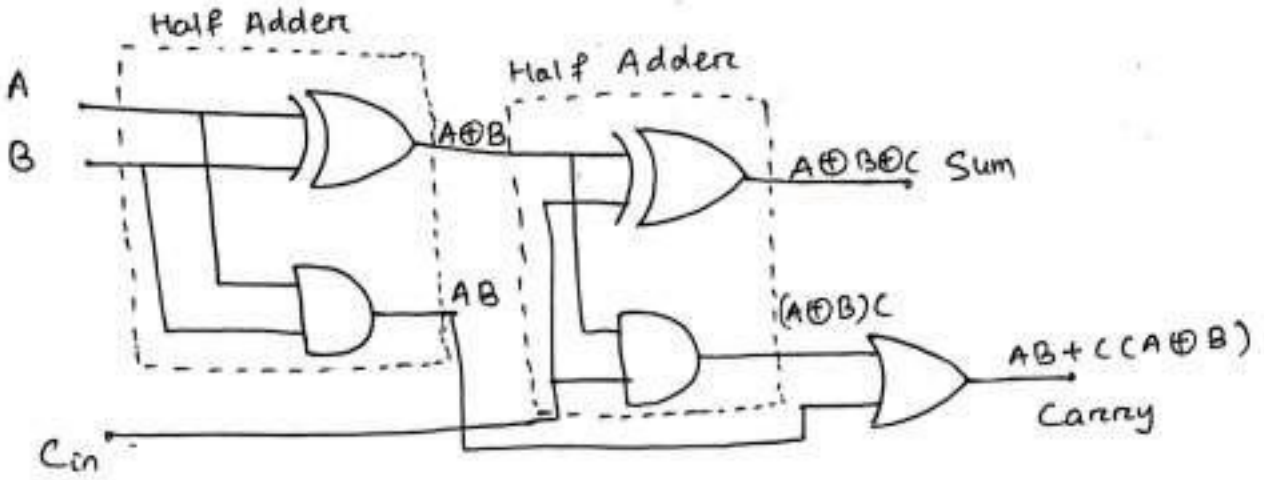
carry (Cout)

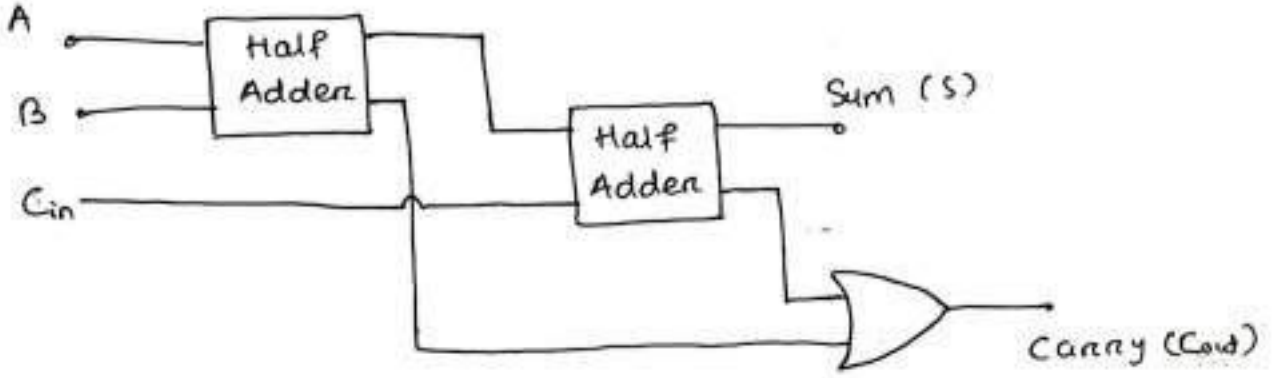
$C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$= C(\bar{A}B + A\bar{B}) + AB(\bar{C} + C)$

$= C(A \oplus B) + AB$

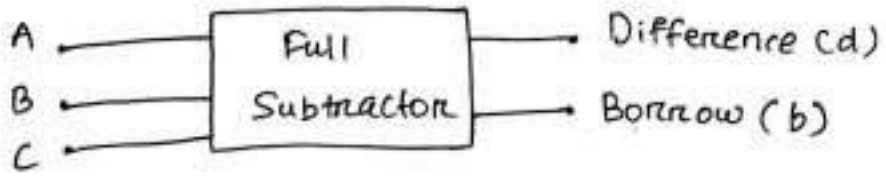
Full Adder using two Half adders & an OR-gate





Full Subtractor

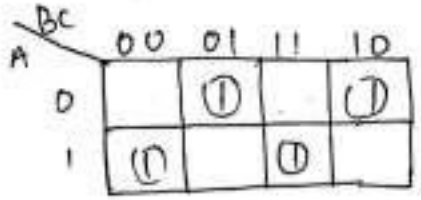
- A full subtractor is an arithmetic circuit which performs a subtraction between two bits taking into account that a '1' may have been borrowed by a lower significant stage.
- A full subtractor has three inputs & two outputs.



Truth Table

Input			output	
A	B	C	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Difference (d)



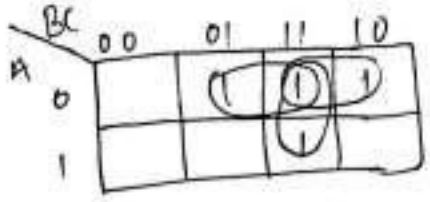
$$d = A\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C}$$

$$= \bar{B}(A\bar{C} + \bar{A}C) + B(AC + \bar{A}\bar{C})$$

$$= \bar{B}(A \oplus C) + B(\overline{A \oplus C})$$

$$= A \oplus B \oplus C$$

Borrow (b)



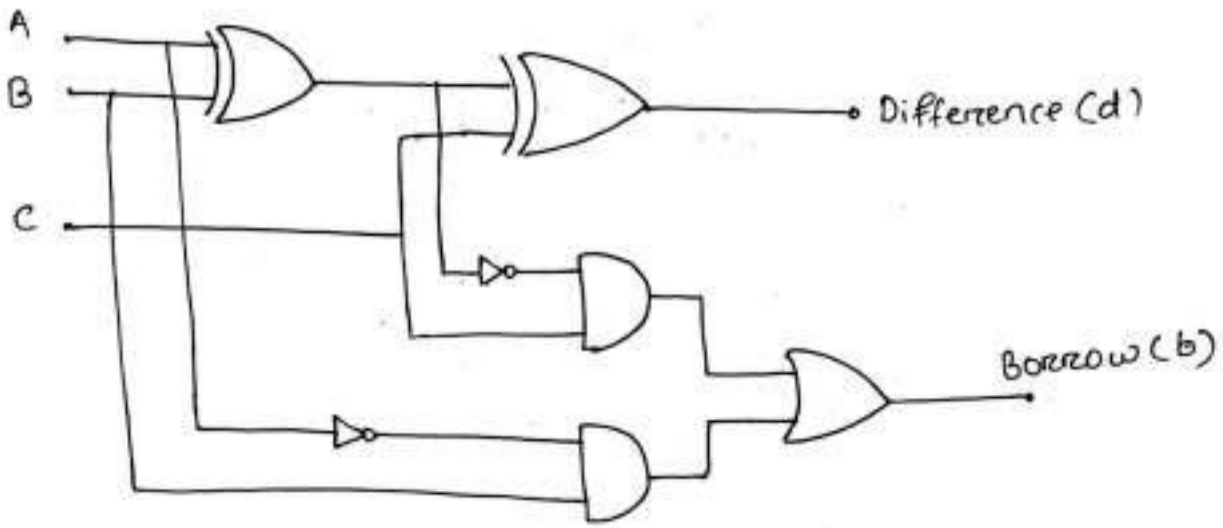
$b = \bar{A}C + BC + \bar{A}B$ (OR)

$$b = \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} + ABC$$

$$= C(\bar{A}\bar{B} + AB) + \bar{A}B(C + \bar{C})$$

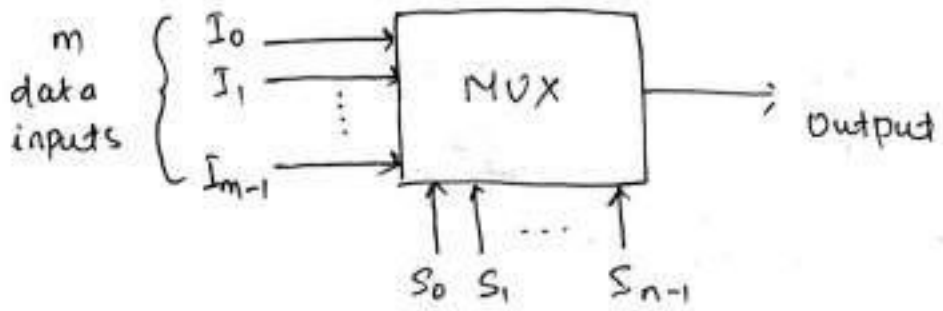
$$= C(\overline{A \oplus B}) + \bar{A}B$$

Logic Diagram



Multiplexer (MUX)

- It is a combinational circuit that selects binary information from one of many input lines and directs it to single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- There are 2^n input lines & n selection lines.
- Multiplexer is also known as Data Selector, many to one circuit, universal logic convertor, parallel to serial converter.

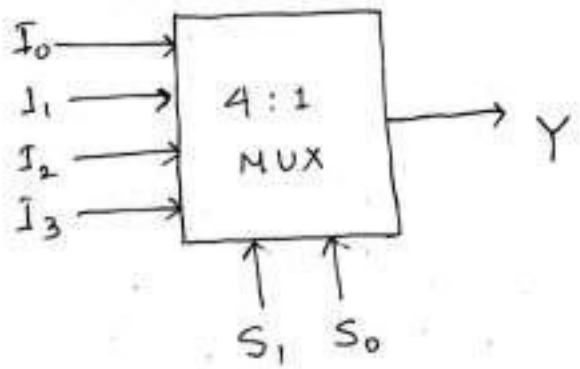


m → Total no. of data input

n → Number of selection lines

4 x 1 Multiplexer

- Each of the four inputs I_0, I_1, I_2 & I_3 are applied to the input of MUX & logic levels applied to the selection lines S_0 & S_1.



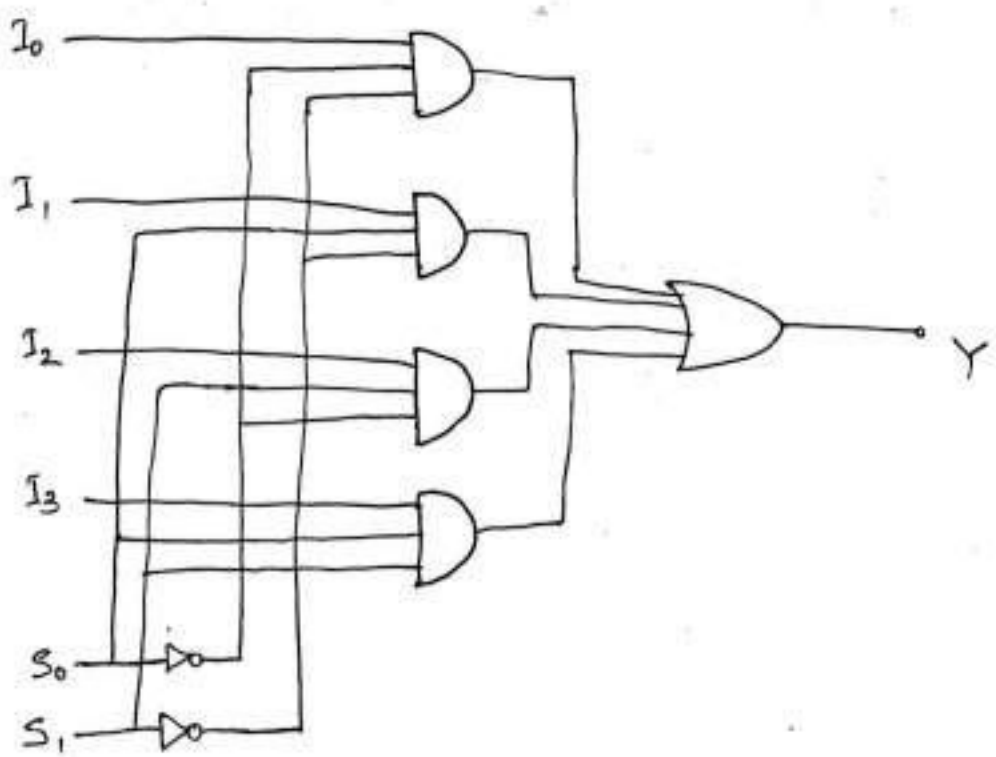
(Block Diagram)

Function Table

S_1	S_2	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$$

Logic circuit

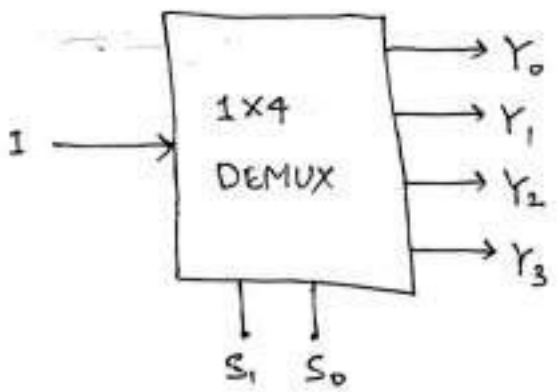


Demultiplexer (DEMUX)

- A demultiplexer is a combinational circuit that receives information on a single line & transmits this on one of 2^n possible output lines.
- The selection of a specific output line is controlled by the bit values of n selection lines.
- Demultiplexer is also known as data distributor, serial to parallel converter, one to many circuit.
- It is used to perform the reverse operation of MUX.

1X4 Demux

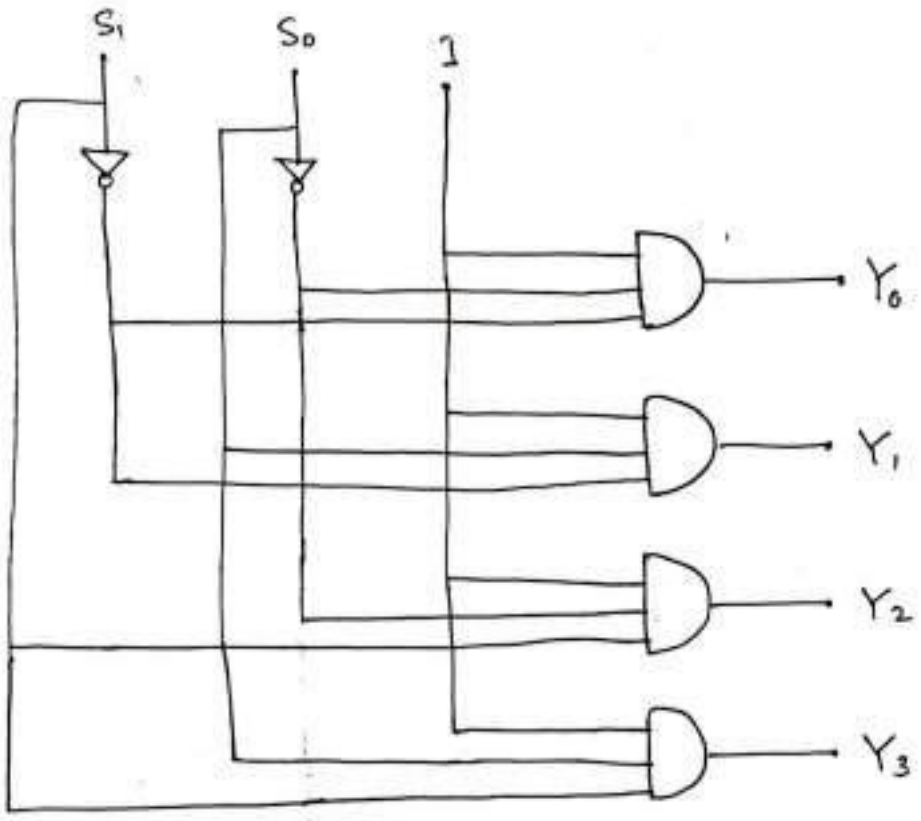
Two selection lines S_0 & S_1 enable only one gate at a time & the data appearing on the input line will pass through the selected gate to the associated output line.



Function Table

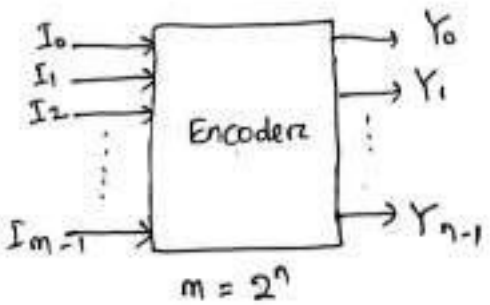
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Logic Diagram



Encoder

- It has 2^n input lines & n output lines.
- Out of 2^n input lines only one is activated at a given time & produces an n-bit output code, depending upon which input is activated.

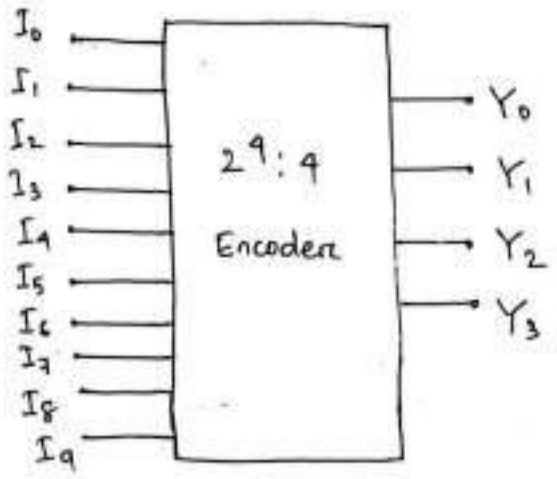


Encoder is used to convert other codes to binary

1. Octal to binary encoder (8 X 3 line)
2. Decimal to BCD encoder (10 X 4 line)
3. Hexadecimal to binary encoder (16 X 4 line)

Decimal to BCD Encoder

- It has ten inputs (0 to 9), & four outputs corresponding to BCD codes.



(Block Diagram)

Truth Table

I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

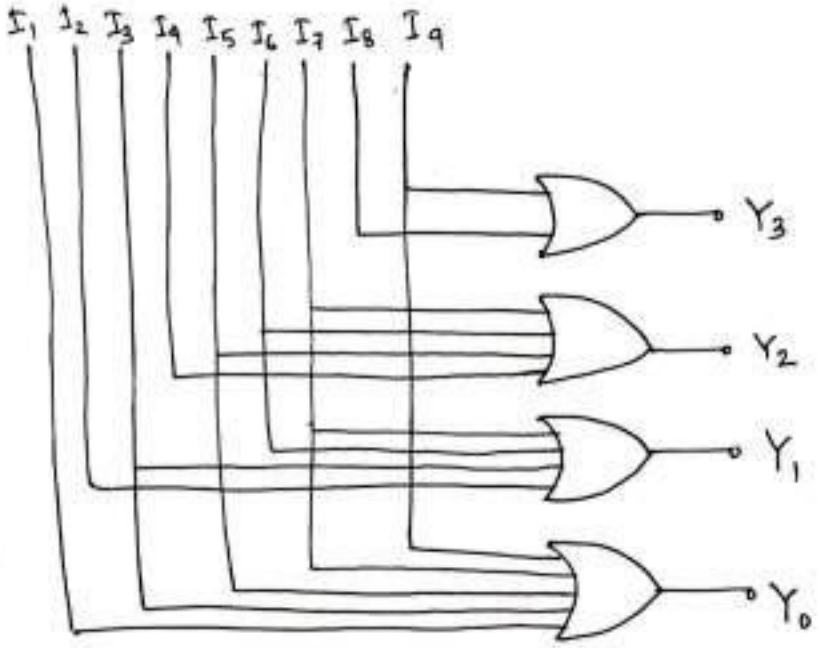
$$Y_0 = I_1 + I_3 + I_5 + I_7 + I_9$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

$$Y_2 = I_4 + I_5 + I_6 + I_7$$

$$Y_3 = I_8 + I_9$$

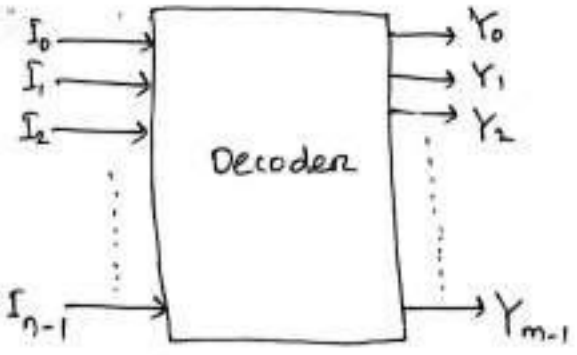
Logic Diagram



Decoder

- It is a combinational logic circuit that converts binary information from 'n' bit input lines to a maximum 2^n unique output lines
- only one output line is activated for each one of possible combinations of input.
- Decoders are used to convert :-

1. Binary to octal (3×8 decoder)
2. Binary to Hexadecimal (4×16 decoder)
3. BCD to decimal (4×10 decoder)



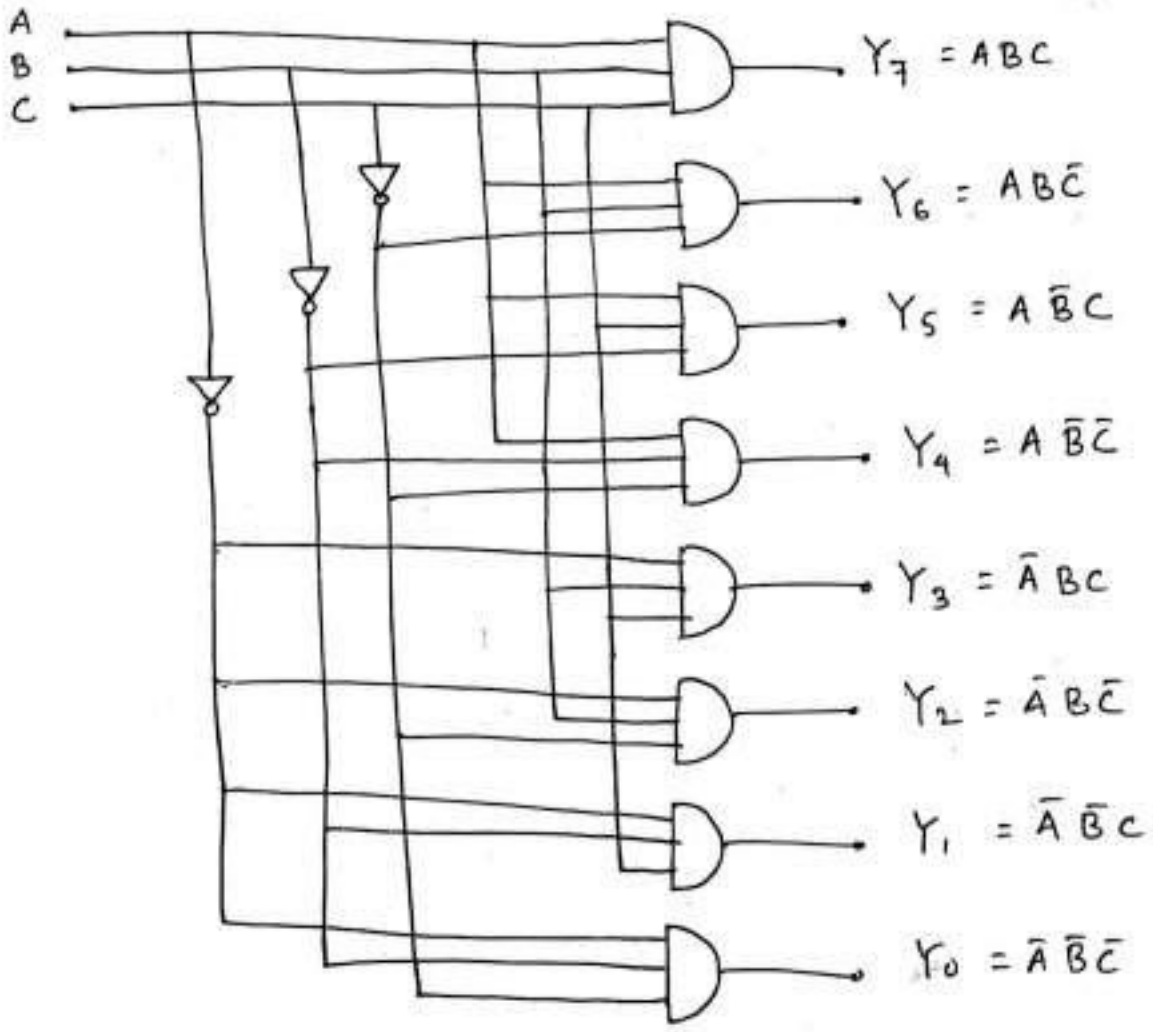
$m = 2^n$

3 x 8 Decoder

Truth Table

A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic Diagram



(3 x 8 Decoder)

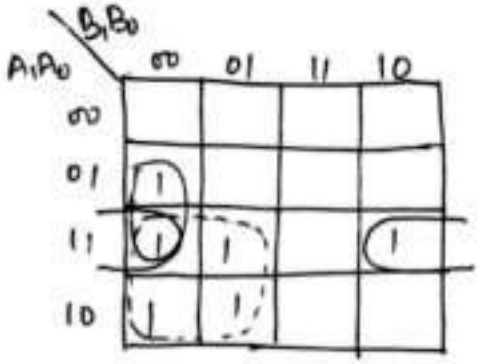
2-bit magnitude Comparator

- A comparator used to compare two binary numbers, each of two bits is called a 2-bit magnitude comparator.
- It consists of four inputs & three outputs to generate less than, equal to, & greater than between two binary numbers.

Truth Table

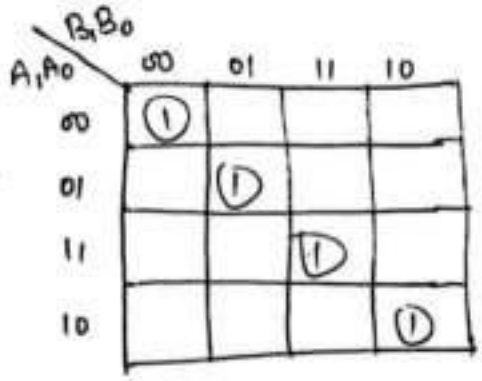
Input				output		
A_1	A_0	B_1	B_0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

A > B



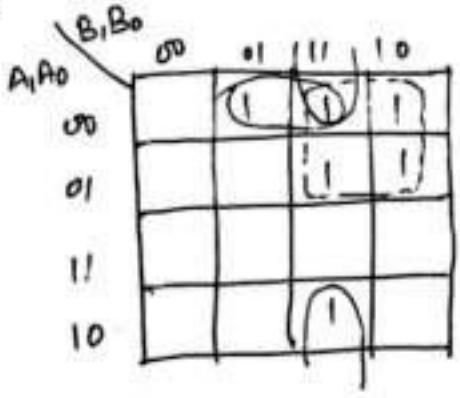
$$A > B = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$$

A = B

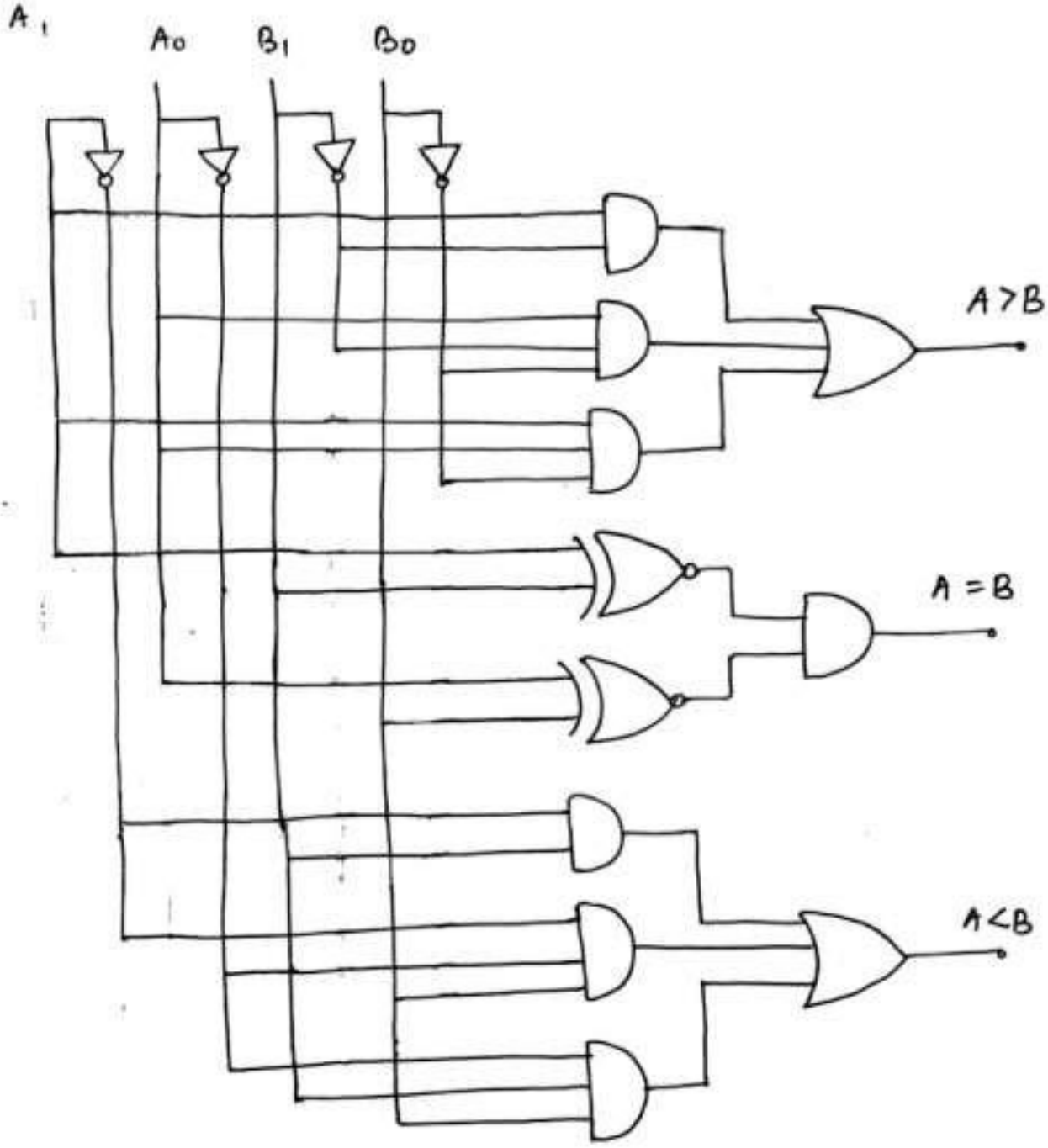


$$A = B = A_1 \cdot$$
$$= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(\bar{A}_0\bar{B}_0 + A_0B_0)$$
$$= (\bar{A}_0\bar{B}_0 + A_0B_0)(\bar{A}_1\bar{B}_1 + A_1B_1)$$
$$= (A_0 \odot B_0)(A_1 \odot B_1)$$

A < B

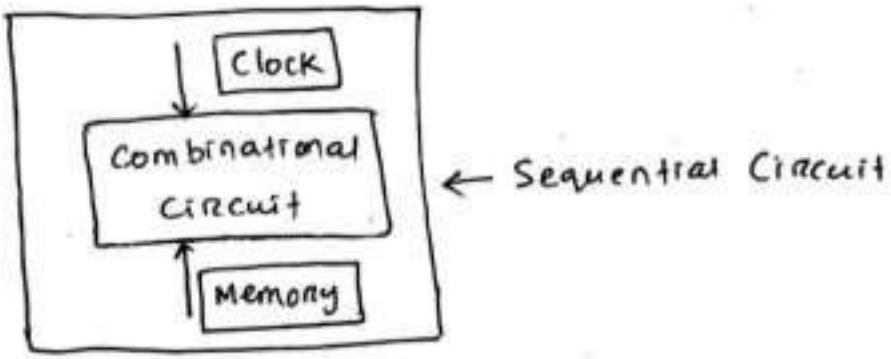


$$A < B = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$$


A1
A0
B1
B0
A>B
A=B
A<B

Sequential Logic Circuit

- In a sequential circuit, the output is dependent upon the present inputs as well as the past inputs & outputs.
- The sequential circuits include the memory elements, which store the past inputs & outputs.
- All the sequential circuits are clocked circuits that is all sequential circuits work with a clock pulse.
- In sequential circuit the output is a function of the present inputs as well as the past inputs & outputs.



- In this memory is used to store previous output & clock pulse decides the amount of time required to get the output.

Difference between combinational & Sequential circuit

Combinational Circuits	Sequential Circuits
- Outputs depend only on present inputs.	- Outputs depend on both present inputs and present.
- Memory elements are not required.	- Memory elements are required.
- Clock signal is not required.	- Clock signal is required.
- Feedback path is not present.	- Feedback path is present.
- Combinational circuits are faster.	- Sequential circuits are slower.
- They are easy to design.	- They are difficult to design.

- The sequential circuits are classified as synchronous sequential circuits & asynchronous sequential circuits depending on the timing of their signals.

Synchronous Sequential circuit

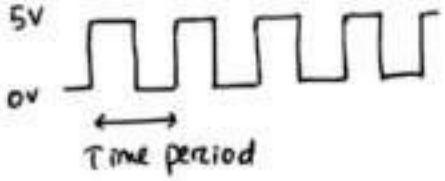
- The change in input signals can affect memory element upon activation of clock signals.
- The maximum operational speed of clock depends on time delays involved.
- In this circuit, memory elements are 'clocked flip-flops'.
- It is easier to design.
- It is generally 'edge triggered'.

Asynchronous Sequential Circuit

- The change in input signals can affect memory element at any instant of time.
- Because of absence of clock, this circuit can operate faster than synchronous circuit.
- In this circuit, memory elements are either unclocked flip flops or time delay elements.
- More difficult to design.
- It is generally level triggered.

Clock Signal

Clock Signal is a periodic signal & its ON time & OFF time need not be the same. we can represent the clock signal as a square wave, when both its ON time & OFF time are same.



- In this case, the time period will be equal to either twice of ON time or twice of OFF time.

- The reciprocal of the time period of clock signal is known as the frequency of the clock signal.
- All the sequential circuits are operated with clock signal. So, the frequency at which the sequentral circuits can be operated accordingly the clock signal frequency has to be chosen.

Types of Triggering

- level triggering
- Edge triggering

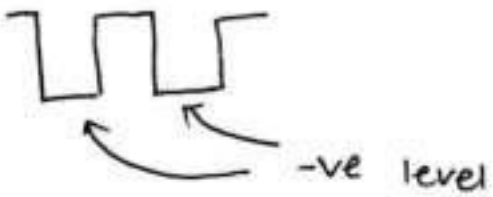
level triggering

There are two levels, namely logic high & Logic low in clock signal. following are the two types of level triggering.

- positive level triggering
- Negative level triggering

Positive level triggering :- If the sequentral circuit is operated with the clock signal when it is in logic high, then that type of triggering is known as positive level triggering.

Negative level triggering :- If the sequentral circuit is operated with the clock signal when it is in logic low, then that type of triggering is known as negative level triggering.

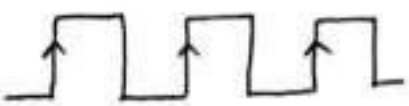


Edge triggering

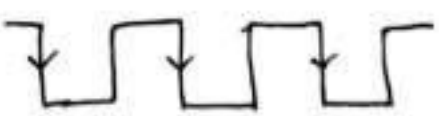
There are two types of transitions that occur in the clock signal. That means, the clock signal transitions either from logic low to logic high (or) logic high to logic low.

- There are two types of edge triggering based on the transitions of clock signal.
- positive edge triggering
- Negative edge triggering

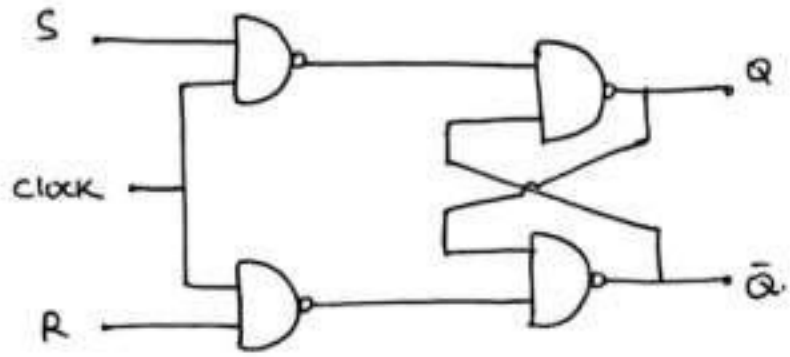
Positive edge triggering :- If the sequential circuit is operated with the clock signal that is transitioning from logic low to logic high, then that type of triggering is known as positive edge triggering. It is also called as rising edge triggering.



Negative edge triggering : If the sequential circuit is operated with the clock signal that is transitioning from logic high to logic low, then that type of triggering is known as negative edge triggering. It is also called as falling edge triggering.



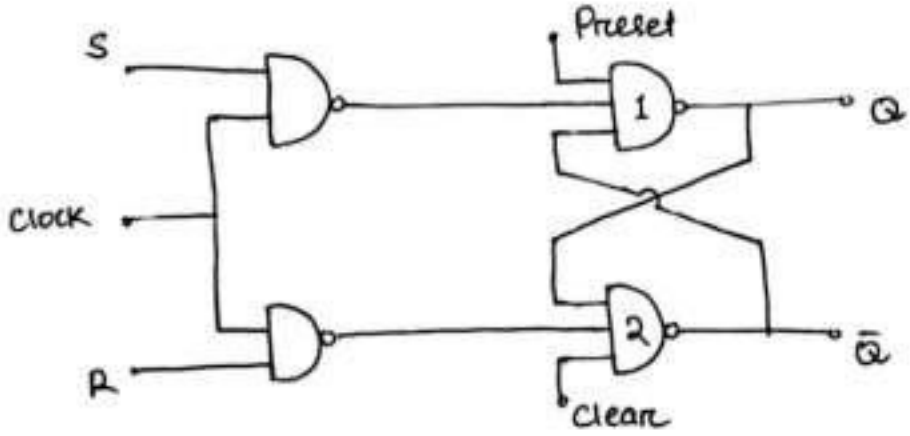
S-R Flip-Flop



Truth Table

Clock	S	R	Q_{n+1}	State
0	X	X	Q_n	
1	0	0	Q_n	Hold
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	X	Invalid

Clocked S-R Flip Flop with preset & clear

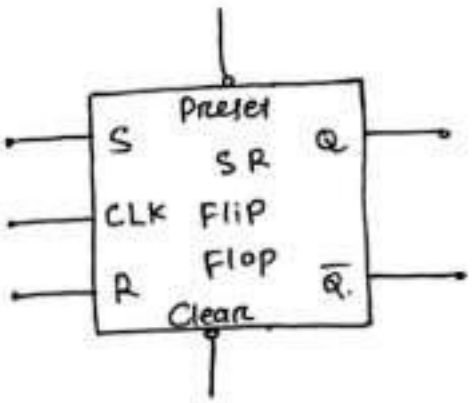


- When preset = 0 & clear = 1, then the output of Gate-1 is 1 & that of Gate-2 is 0, which is independent of inputs S & R, and the flip-flop is Set.
- When preset = 1 & clear = 0, then the output of Gate-2 is 1 which makes the output of Gate-1 is 0 (i.e. Q=0), which is independent of inputs S & R, and the flip flop is reset.

- When preset = 1 & clear = 1, then outputs of Gate-1 & Gate-2 depend on other inputs. For normal operation, preset & clear are connected to logic 1.
- When preset = 0 & clear = 0, then outputs of Gate-1 & Gate-2 try to become 1. Here, the uncertain state occurs & hence, preset = 0 & clear = 0 is not used.

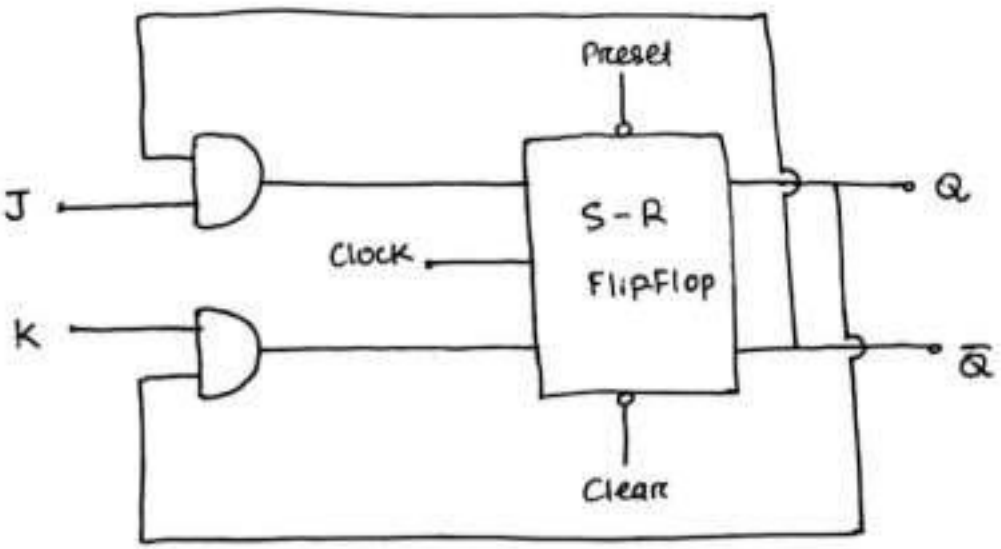
Input				Output
S	R	Preset	Clear	Q_{n+1}
X	X	0	1	1
X	X	1	0	0
0	0	1	1	Q_n
0	1	1	1	0
1	0	1	1	1
1	1	1	1	Invalid

(Truth Table)



(Symbol)

JK Flip Flop using S-R flip-flop



- The uncertainty in the state of an S-R flip flop when S = R = 1 can be eliminated by converting it into a J-K Flip-Flop.

$$S = J \cdot \bar{Q}$$

$$R = K \cdot Q$$

Inputs		outputs		Inputs to SR		output	
J	K	Q_n	$\bar{Q}_n$	S	R	Q_{n+1}	
0	0	0	1	0	0	0	Q_n
0	0	1	0	0	0	1	
1	0	0	1	1	0	1	1
1	0	1	0	0	0	1	
0	1	0	1	0	0	0	0
0	1	1	0	0	1	0	
1	1	0	1	1	0	1	$\bar{Q}_n$
1	1	1	0	0	1	0	

Truth Table of JK FlipFlop

Clock	J	K	Q_{n+1}	
0	x	x	Q_n	→ memory
1	0	0	Q_n	→ Hold
1	0	1	0	→ Reset
1	1	0	1	→ Set
1	1	1	$\bar{Q}_n$	→ Toggle

Race Around Condition

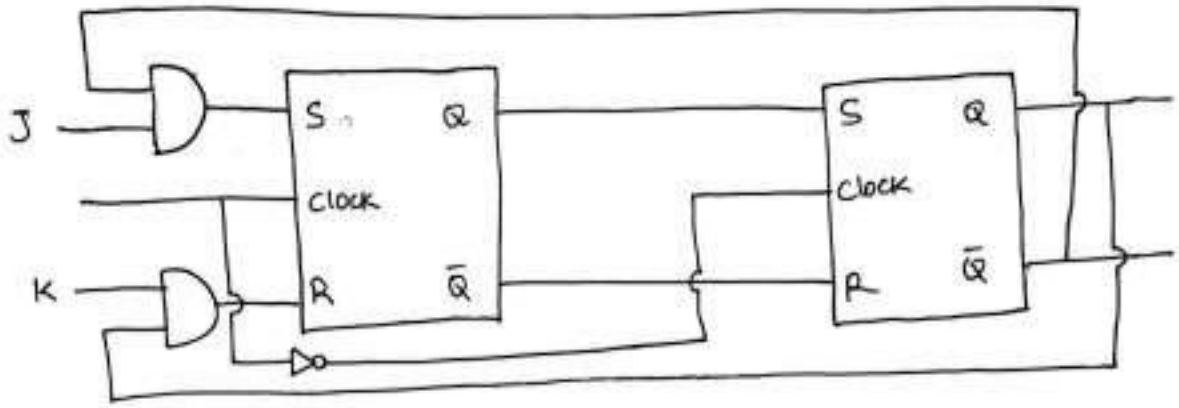
- If output at the FlipFlop is toggled more than once for one clock completion then it is known as Race around condition or Racing.
- SR & D Flip Flops are free from race condition.
- Edge triggering is also free from race condition.
- Race condition may be occured only in level triggering JK or T-Flip Flops.
- To avoid race condition, there are two methods :-

① To maintain $t_{pw} < t_{pd(FF)} < T$

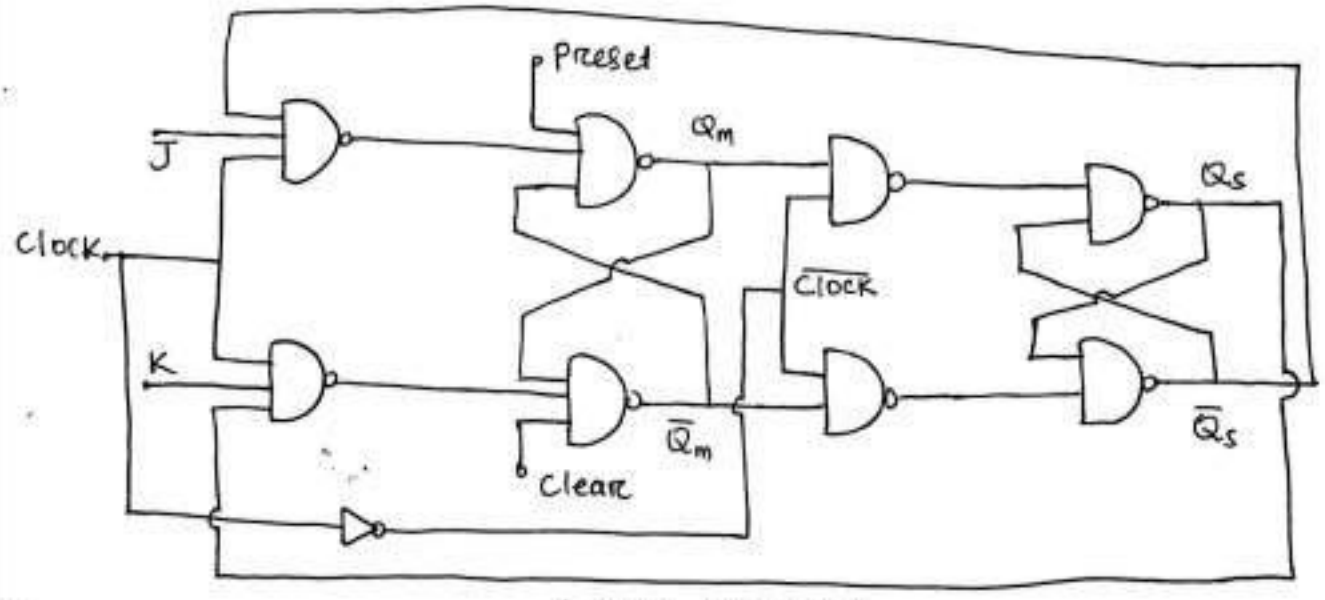
② Master Slave J-K Flip Flop

[T - time period, t_{pw} - pulse width, $t_{pd(FF)}$ - Propagation delay of FF]

Master Slave JK Flip Flop



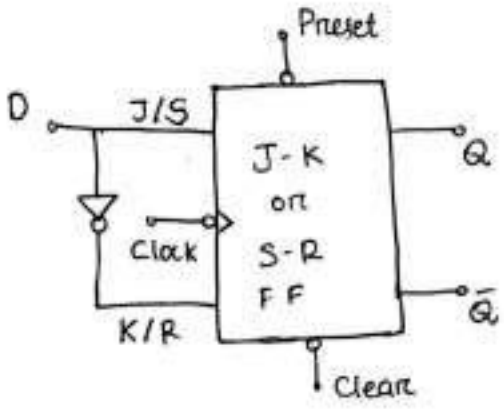
- The output of second S-R Flipflop is given to the input of the first SR flip-Flop through AND gate.
- The clock is directly applied to the first S-R Flip Flop after the NOT is applied to the second SR Flip Flop.
- When clock = 1, the first Flip Flop is enabled & the second flip Flop is disabled.
- Since the second Flip Flop is disabled, the output cannot be changed during the clock, which is the input for the first flip flop. Hence, inputs are not changed during the clock & the problem of race-around condition is resolved.
- When clock = 0, the first flip flop is disabled & the second Flip-flop is enabled. The output of first Flip-Flop is the input for the Second Flip-Flop.
- The first Flip Flop is known as Master & the second is known as Slave.

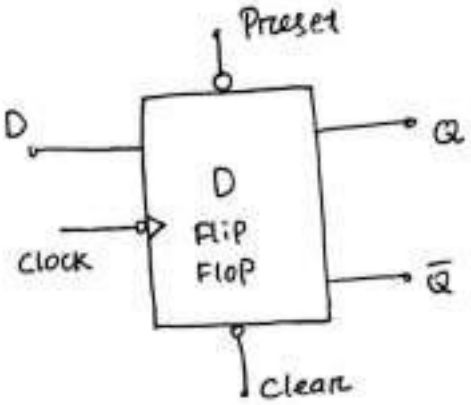


(logic Diagram)

D - Flip Flop

It has only one input referred to as D-input or data input.





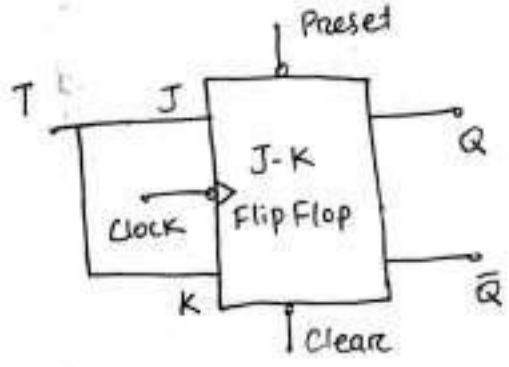
Truth Table

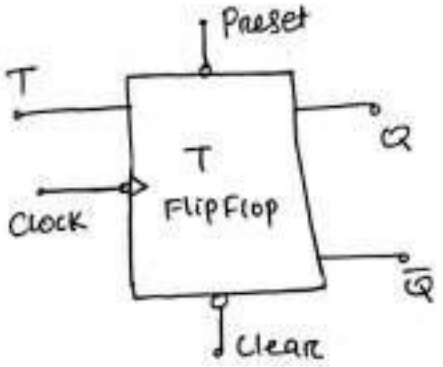
Clock	D	Q_{n+1}
0	X	Q_n
1	0	0
1	1	1

- The input data appears at the output at the end of the clock pulse. Thus the transfer of data from the input to the output is delayed & hence the name delay (D) Flip Flop.

T - Flip Flop

- In a J-K Flip Flop, if J=K, the resulting Flip Flop is referred to as a T-type Flip Flop.
- It has only one input, referred to as T-input.





Truth Table

Clock	T	Q_{n+1}
0	X	Q_n
1	0	Q_n
1	1	$\bar{Q}_n$

- The designation T comes from the ability of the flip flop to Toggle or change state.

Applications of Flip-Flops

(i) Serial & parallel data storage
(ii) Data transfer
(iii) Serial to parallel converter
(iv) Parallel to serial converter
(v) Latch
(vi) Counters
(vii) Frequency division

Modulus of a Counter

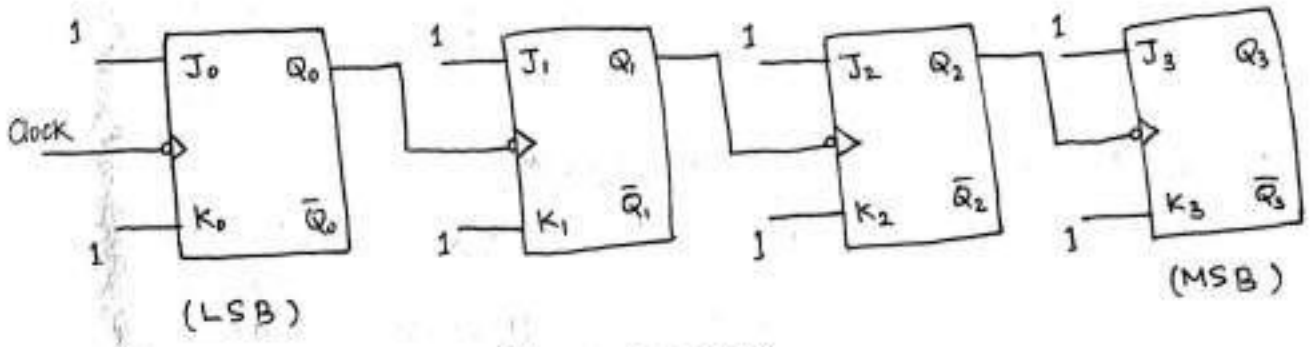
- modulus counter or simply MOD counter are defined based on the number of states that the counter will sequence through before returning back to its original value.
- ex - a 2-bit counter that counts from 00_2 to 11_2 in binary has a modulus value of 4 ($00 \rightarrow 01 \rightarrow 10 \rightarrow 11$). Therefore it is called a modulo-4 or mod-4 counter.
- Mod-K up counter can count K number of states from 0 to K-1.

Difference between Synchronous Counter & Asynchronous Counter

Synchronous Counter	Asynchronous Counter
1. All the flip flops are triggered simultaneously with the same clock.	1. Different FlipFlops are triggered with different clock.
2. Operation is faster.	2. Operation is slower.
3. Any required sequence can be designed.	3. Will operate only in a fixed count sequence.
4. Designing is complex as the number of states increases.	4. Designing is easy even for more number of states.
5. ex- Ring Counter, Johnson Counter	5. ex- Ripple UP counter, Ripple DOWN Counter.

4-bit Asynchronous Counter

- It consists of a series connection of complementing J-K flip-flops, with the output of each flipflop connected to the clock pulse input of the next higher order flip flop.
- The flip flop holding the LSB receives the incoming clock pulses.



(logic diagram)

Operation

(i) Here negative edge triggering is applied. So Flip Flops change their State on the negative going edge of clock pulse.

(ii) Note that all the flip flops are considered in toggled mode.

(iii) Q_0 will change its state in every clock pulse.

(iv) Q_1 will change its State when Q_0 will change from 1 to 0.

(v) Q_2 will change its State when Q_1 will change from 1 to 0.

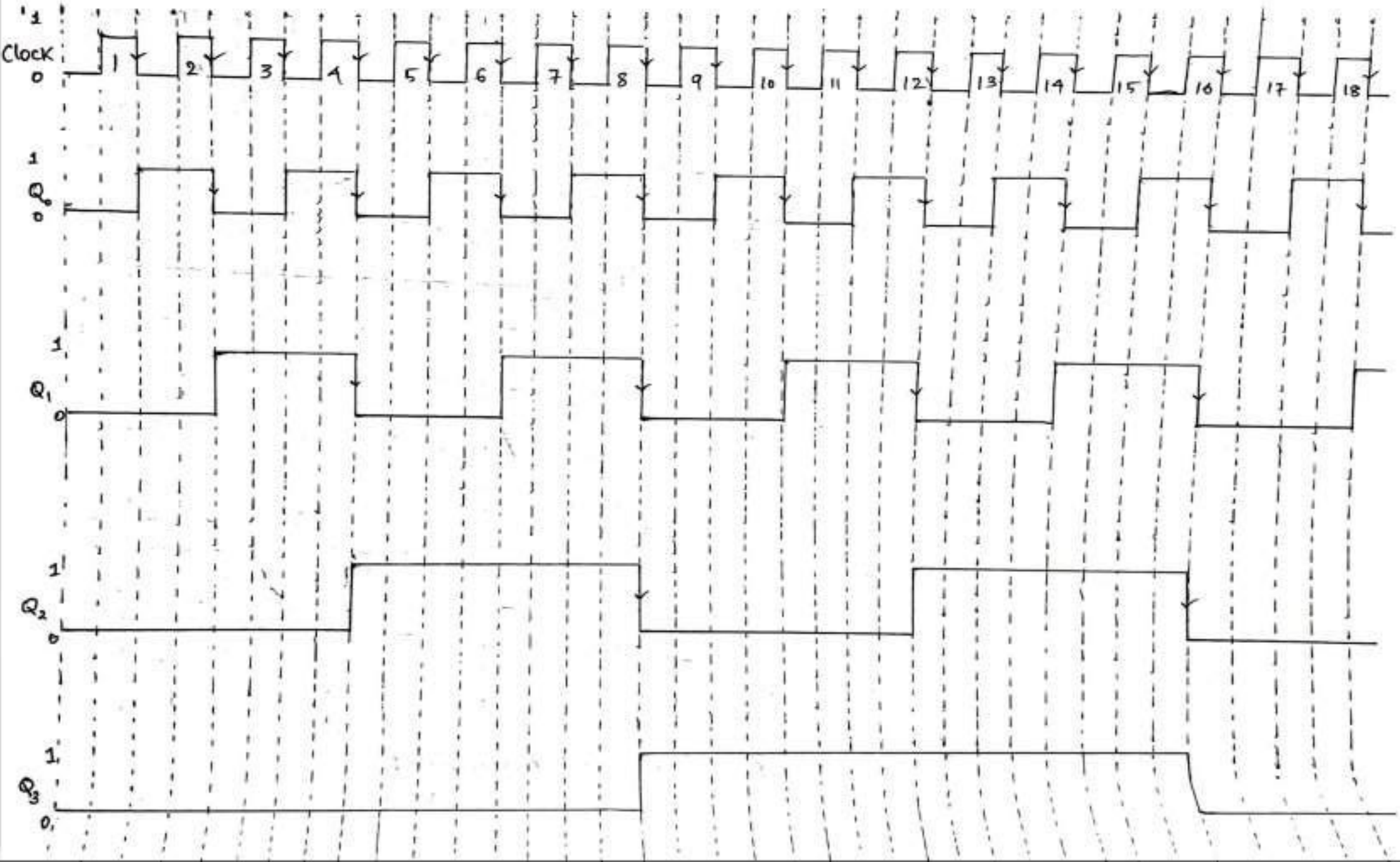
(vi) Q_3 will change its state when Q_2 will change from 1 to 0.

Truth Table

Clock	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

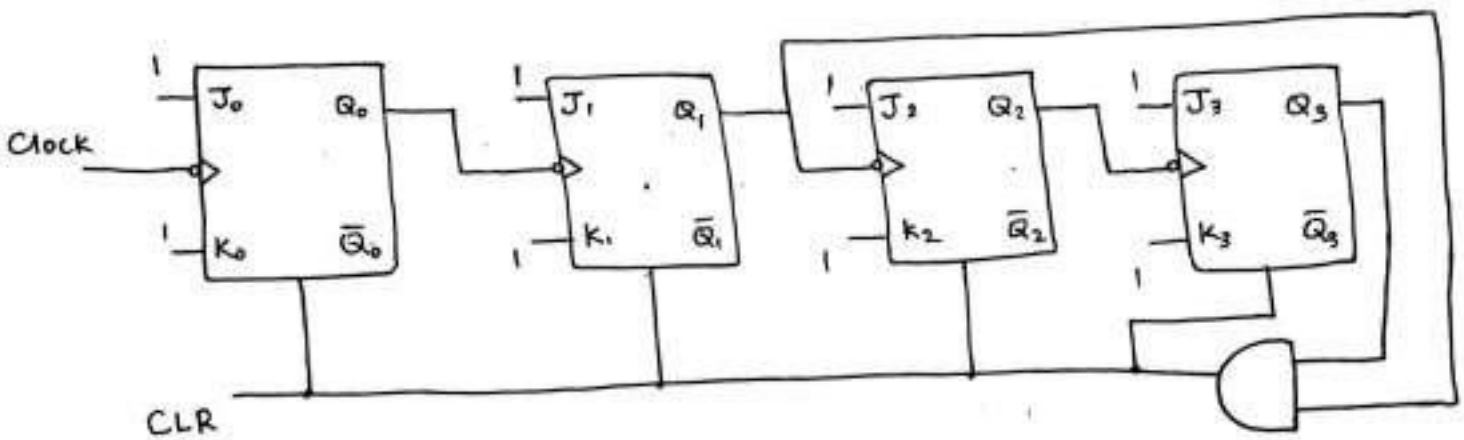
- Initially all flip flops are set to zero.
- The maximum possible state = $2^4 = 16$ (from 0 to 15)
- If the input clock frequency is f then the output frequency is $f/16$.
- It is a 4 bit up counter, which counts from 0 to 15.
- For 4 bit down counter, positive edge triggering is applied & clock is given from Q (or) negative edge triggering is applied & clock is given from $\bar{Q}$.
- Four bit down counter counts from 15 to 0.

Timing Diagram



Asynchronous Decade Counter

- Here the total number of flip flops required is 4, thus the number of used state = 10 & the number of unused state = 6.
- CLR is used to clear the flip flop i.e 0.
- In order to design a non binary decade counter a logic gate is used which detects 10 state from 0000 to 1001 & as soon as 1010 appears it clears all the flip-flops.



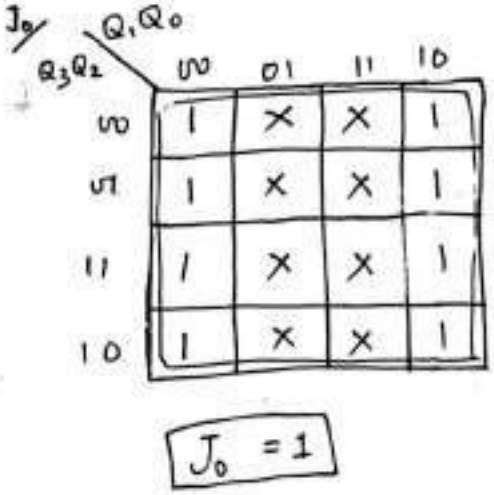
Truth Table

Clock	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0

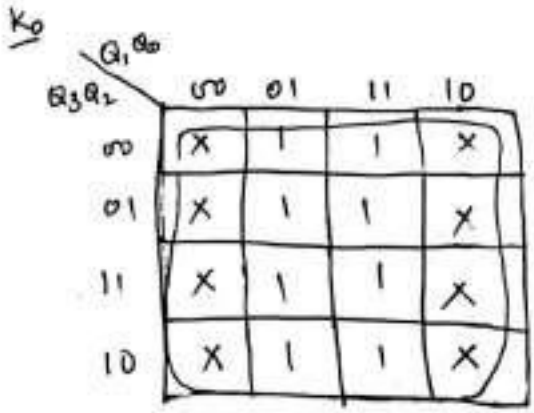
4-bit Synchronous Counter

- In Synchronous Counter all the flipflops given clock simultaneously.

Present State				Next State				Flip Flop Inputs					
Q_3	Q_2	Q_1	Q_0	Q_3'	Q_2'	Q_1'	Q_0'	J_0	K_0	J_1	K_1	J_2	K_2	J_3	K_3
0	0	0	0	0	0	0	1	1	X	0	X	0	X	0	X
0	0	0	1	0	0	1	0	X	1	1	X	0	X	0	X
0	0	1	0	0	0	1	1	1	X	X	0	0	X	0	X
0	0	1	1	0	1	0	0	X	1	X	1	1	X	0	X
0	1	0	0	0	1	0	1	1	X	0	X	X	0	0	X
0	1	0	1	0	1	1	0	X	1	1	X	X	0	0	X
0	1	1	0	0	1	1	1	1	X	X	0	X	0	0	X
0	1	1	1	1	0	0	0	X	1	X	1	X	1	1	X
1	0	0	0	1	0	0	1	1	X	0	X	0	X	X	0
1	0	0	1	1	0	1	0	X	1	1	X	0	X	X	0
1	0	1	0	1	0	1	1	1	X	X	0	0	X	X	0
1	0	1	1	1	1	0	0	X	1	X	1	1	X	X	0
1	1	0	0	1	1	0	1	1	X	0	X	X	0	X	0
1	1	0	1	1	1	1	0	X	1	1	X	X	0	X	0
1	1	1	0	1	1	1	1	1	X	X	0	X	0	X	0
1	1	1	1	0	0	0	0	X	1	X	1	X	1	X	1
0	0	0	0												



$J_0 = 1$



$K_0 = 1$

J_1 /

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00		1	X	X
01		1	X	X
11		1	X	X
10		1	X	X

$J_1 = Q_0$

K_1 /

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	1	
01	X	X	1	
11	X	X	1	
10	X	X	1	

$K_1 = Q_0$

J_2 /

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00			1	
01	X	X	X	X
11	X	X	X	X
10			1	

$J_2 = Q_1Q_0$

K_2 /

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01			1	
11			1	
10	X	X	X	X

$K_2 = Q_1Q_0$

J_3 /

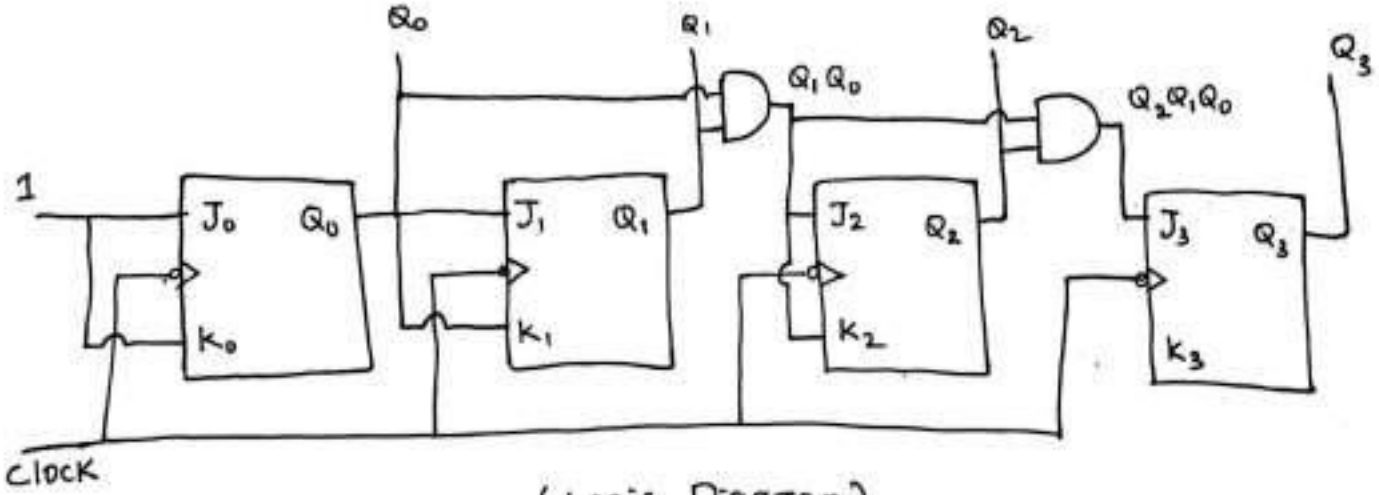
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00				
01			1	
11	X	X	X	X
10	X	X	X	X

$J_3 = Q_2Q_1Q_0$

K_3 /

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11			1	
10				

$K_3 = Q_2Q_1Q_0$



(Logic Diagram)

Registers

- A register is a digital circuit with two basic functions i.e. data storage & data movement.
- It is basically a group of flip-flops logically connected to perform various functions.
- To store a group of N-bit word, the number of flip-flops required is N (one for each bit).
- A register is a group of binary storage cells suitable for holding binary information. In addition to the flip-flops, a register may have combinational gates that perform certain data processing tasks. Thus a register consists of a group of flip-flops & gates that effect their transition.

Shift Register

- A register capable of shifting the binary information entered into it from an external binary source is called the shift register.
- It is a sequential circuit mainly used to store or shift binary data either to the right or to the left.
- All flipflops receive a common clock pulse which causes shift from one state to the next.
- In shift register each clock pulse shifts the contents of register one bit position to the right or left.

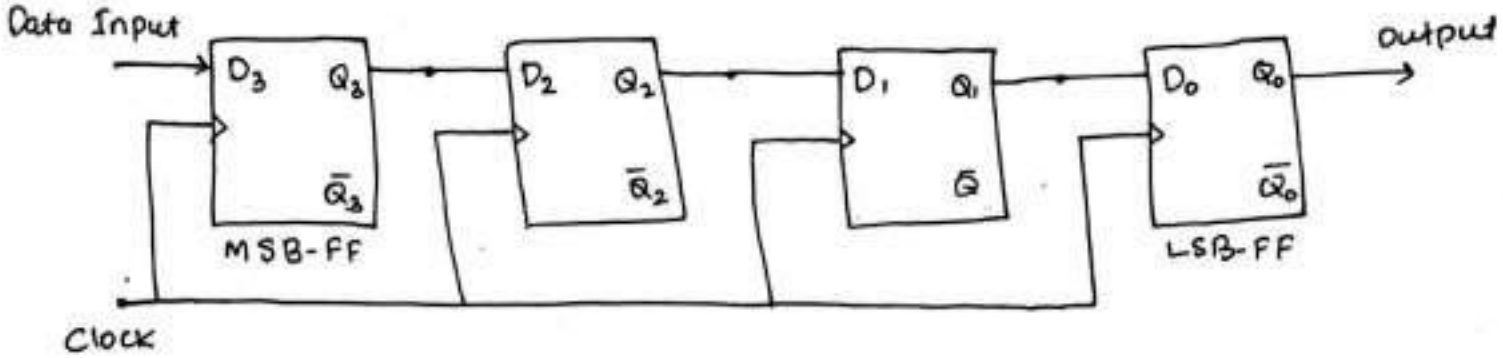
Classification of Shift Registers

(i) Serial IN, Serial OUT shift register (SISO)
(ii) Serial IN, Parallel OUT shift register (SIPO)
(iii) Parallel IN, Serial OUT Shift register (PISO)
(iv) Parallel IN, Parallel OUT Shift register (PIPO)

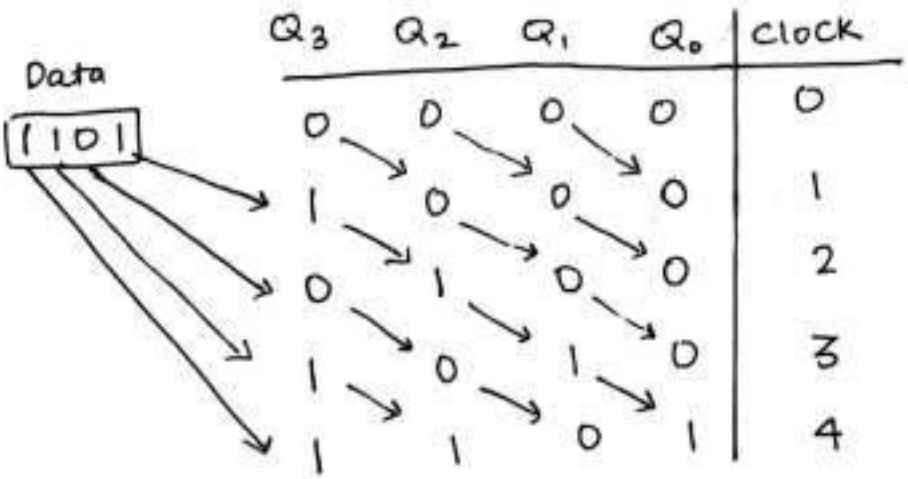
IN - Input
OUT - Output

① Serial IN Serial OUT shift register (SISO)

- The Serial IN Serial OUT shift register accepts the data serially, one bit at a time on a single input line. It produces the stored binary information on its single output line in serial form.



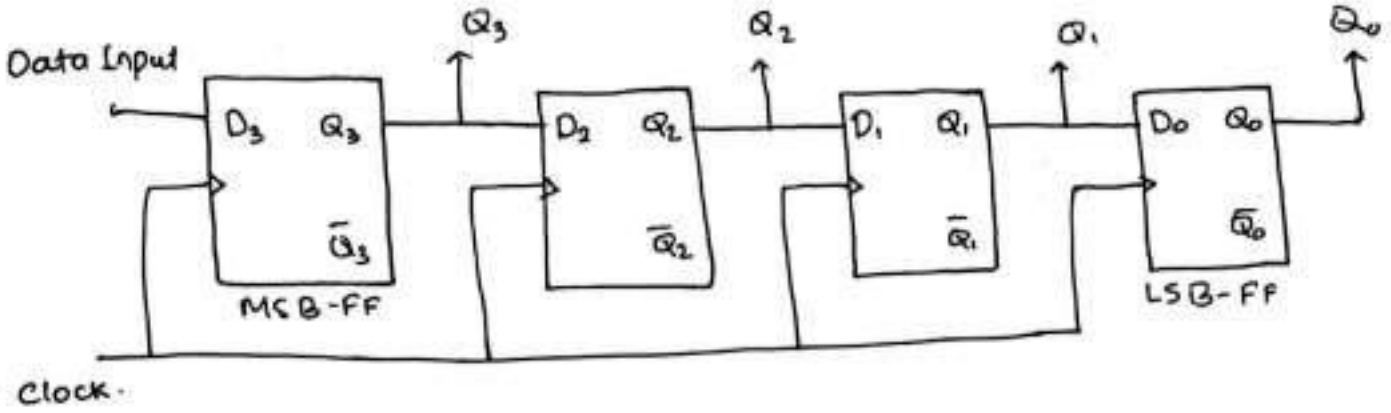
- Let the binary information 1101 is applied to the input.
- Since shift register is reset, so initially all the flip-flop outputs are zero i.e 0000.
- Data 1101 is applied to the input line. Therefore in the right shift SISO register, LSB data is applied at the MSB FF (D_3)



- In 'n' bit register, to enter 'n' bit data, it requires 'n' clock pulses in serial form.
- If 'n' bit data is stored in SISO register then output is taken serially, for this it requires (n-1) clock pulses.
- SISO register is used to provide n clock pulses delay to the input data.

(ii) Serial IN Parallel OUT shift register (SIPO)

- In this shift register, data is entered in serial form but output is taken in parallel form. Therefore, once the data is stored each bit is available on its respective output lines simultaneously.



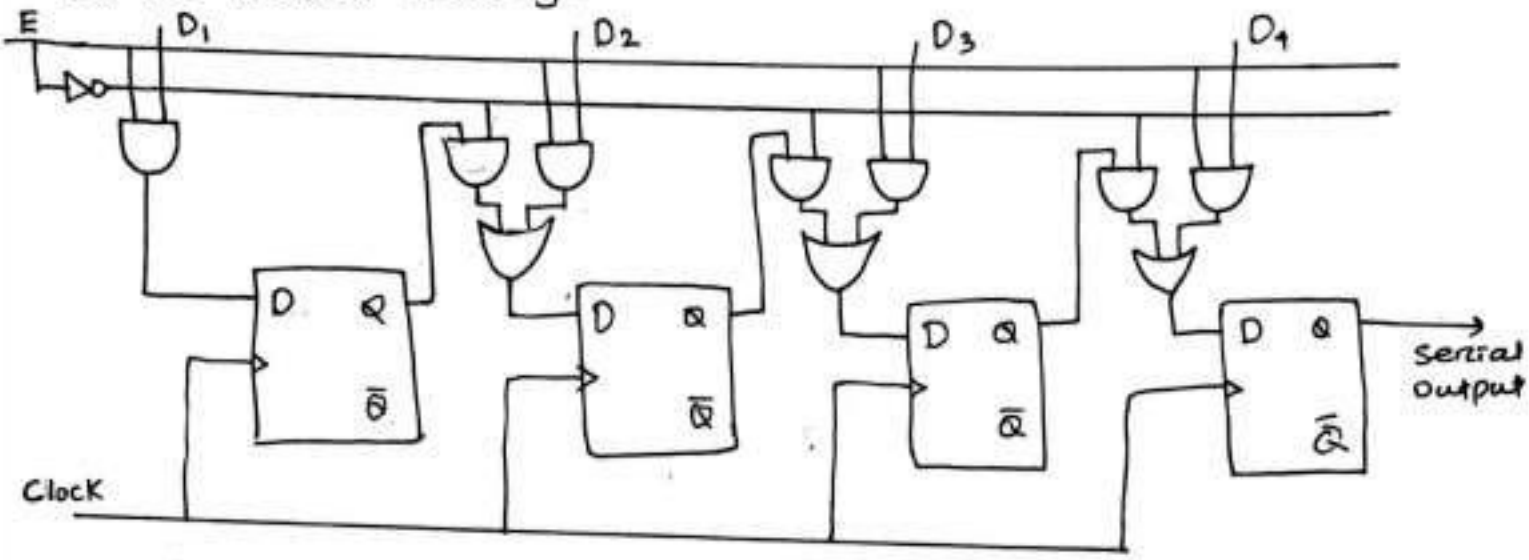
Input Data 1101

Q_3	Q_2	Q_1	Q_0	Clock	
0	0	0	0	0	→ Initially
1	0	0	0	1	
0	1	0	0	2	
1	0	1	0	3	
1	1	0	1	4	

- For storing n-bit serial input data number of clock pulses required = n.
- For n-bit parallel output data to be stored the number of clock pulses required = 0.

(iii) Parallel IN Serial OUT Shift Register (PISO)

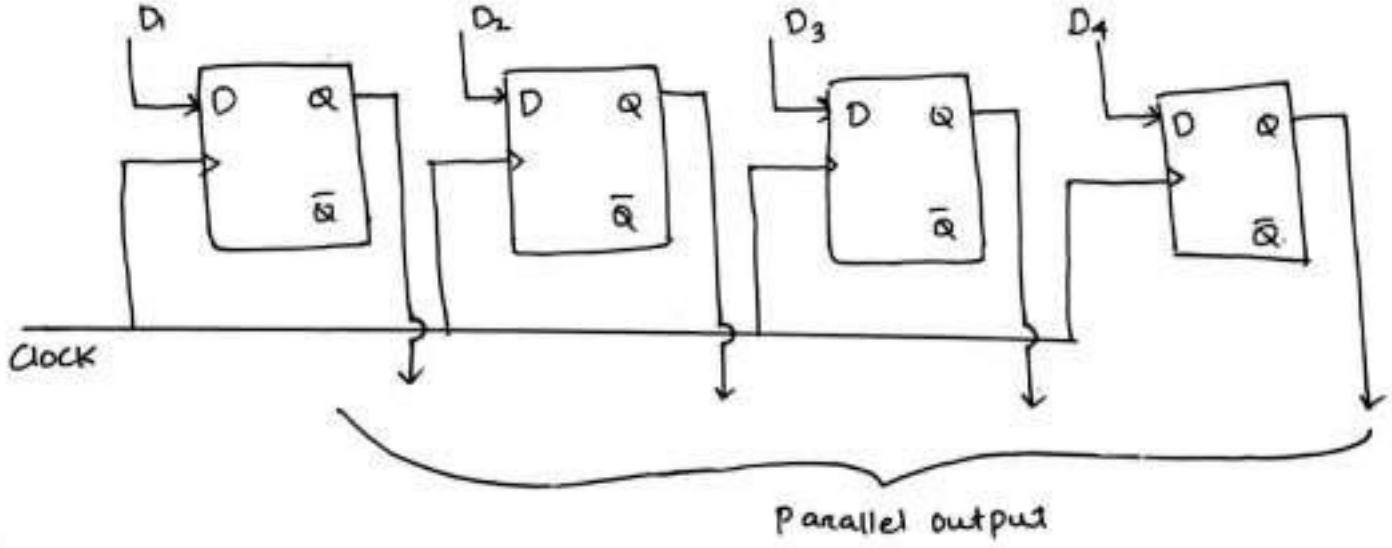
In PISO shift register the data bits are entered simultaneously into the respective flip flops and the shifted data is available at the output serially.



- When E is 'one' input data is given & when E is 'zero' output taken.
- To store parallel data of n-bit it requires 1 clock pulse.
- To store serial output data of n-bit the no. of clock pulses required are (n-1).

(iv) Parallel IN Parallel OUT shift register (PIPO)

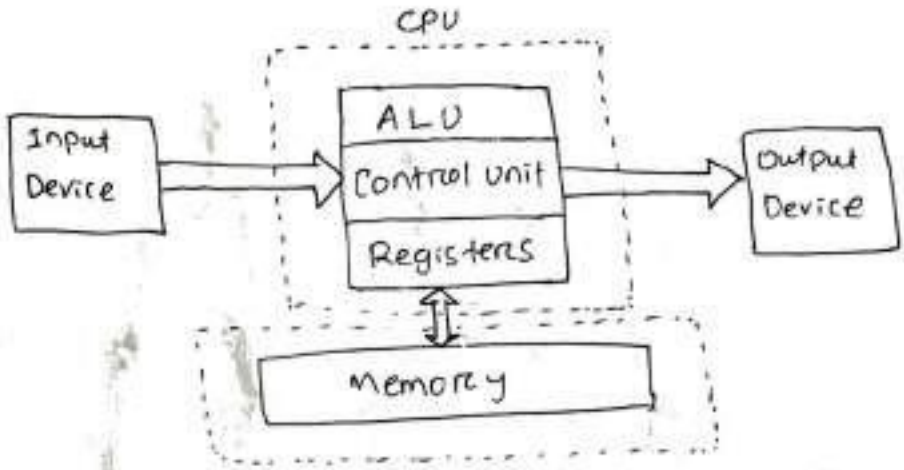
- In this register the input is a parallel entry & the output is also taken simultaneously.



- For parallel IN data the number of clock pulse required is 1
- For parallel OUT data the number of clock pulse required is 0.

Microprocessor

- A microprocessor is a multiple purpose programmable clock driven, register based electronic device that reads binary instructions from memory, accepts binary data as input & processing this data according to the instructions written in the memory.
- The microprocessor is capable of performing computing functions & making decisions to change the sequence of program execution.
- The microprocessor can be embedded in a larger system, & can function as the CPU of a computer called a microcomputer.



(Block Digram of Microcomputer)

- Input device is a device that transfers information from outside world to the computer example: keyboard, mouse, microphone etc.
- The output device transfers information from computer to the outside world like monitor, printers, speaker, projector etc.
- Memory is an electronic medium that stores binary information.
- CPU (central processing unit) is the heard of computer systems. The microprocessor in any microcomputer acts as a CPU. The CPU can be made up with ALU, CP, Registers.
- ALU is the group of circuits that performs arithmetic & logical operations.
- CU (control unit) is a group of circuits that provide timings & signals to all the operations in the computer & controls the data flow.

System Bus

A bus is a group of wires/lines used to transfer data between components inside a computer or between computers. They are communication path used to carry the signals between microprocessor & peripherals.

The system bus of a microprocessor is of 3 types

① Address Bus

- It is a group of lines that are used to send a memory address or a device address from the microprocessor unit (MPU) to the memory or the peripheral.
- The address bus is always uni-directional i.e. address always goes out of the microprocessor.
- If the address lines are n for a MPU then its addressing capacity is 2^n.

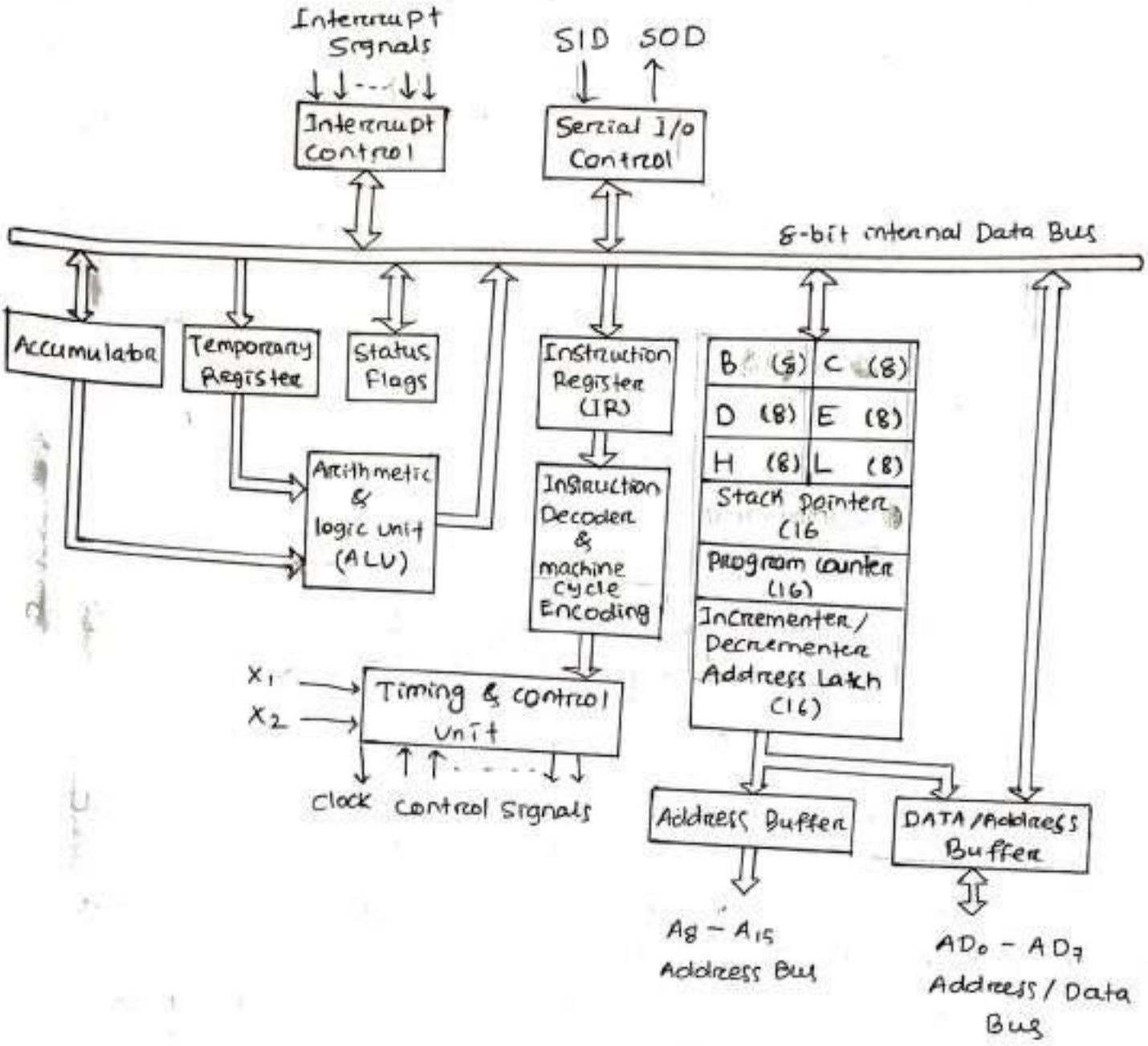
② Data Bus

- It is group of lines used to transfer data between the microprocessor & peripherals and/or memory.
- Data bus is always bi-directional.

③ Control Bus

- Control Bus provides signals to control the flow of data.

Architecture of 8085 Microprocessor



ALU

The arithmetic & logic unit, ALU, performs arithmetic & logical operations such as Addition, subtraction, logical AND, OR, XOR, NOT, increament, decrement, left shift, rotate left, rotate right, clear etc.

Timing & Control unit

- The timing & control unit is a section of the CPU. It generates timing & control signals which are necessary for the execution of instructions.
- It controls data flow between CPU & peripherals.

- It controls the entire operations of the microprocessor & peripherals connected to it. Thus it is seen that the control unit of the CPU acts as the brain of the computer system.

Register

- Registers are used by the microprocessor for temporary storage & manipulation of data & instructions.
- 8085 microprocessor has the following registers:
  - (i) one 8 bit accumulator i.e Register A
  - (ii) Six 8-bit general purpose registers. These are B, C, D, E, H & L
  - (iii) One 16-bit stack pointer, SP
  - (iv) one 16-bit program counter, PC
  - (v) Instruction register
  - (vi) Temporary register
- 8085 microprocessor also contains a set of five flip flops which serve as flags or status flags

Accumulator

- The accumulator is an 8-bit register associated with the ALU. The register A in the 8085 is an accumulator.
- It is used to hold one of the operands of an arithmetic or logical operation. It serves as one input to the ALU.
- The final result of an arithmetic or logical operation is placed in the accumulator.

General purpose Registers

- The 8085 microprocessor contains six 8-bit general-purpose registers. They are B, C, D, E, H & L register.
- To hold 16-bit data a combination of 2 8-bit registers can be employed. The combination of two 8-bit registers is known as a register-pair. The valid register pairs in the 8085 are B-C, D-E & H-L.
- The H-L pair is used to act as memory pointer & for this purpose it holds the 16-bit address of a memory location.

Program Counter (PC)

- It is a 16-bit special purpose register. It is used to hold the memory address of the next instruction to be executed.
- The microprocessor increments the content of the program counter during the execution of an instruction so that it points to the address of the next instruction in the program at the end of the execution of an instruction.

Stack Pointer (SP)

- It is a 16-bit special purpose register. The stack is a sequence of memory locations set aside by a programmer to store/retrieve the contents of accumulation, flags, program counter & general purpose registers during the execution of a program.
- The stack pointer (SP) controls addressing of the stack. The SP holds the address of the top element of data stored in the stack.

Instruction Register

- The instruction register holds the operation code or instruction code of the instruction which is being decoded & executed.

Temporary Register

- It is an 8-bit register associated with the ALU. It holds data during an arithmetic/logical operation.
- It is used by the microprocessor. It is not accessible to programmer.

Flags

- 8085 microprocessor contains five flip flops to serve as status flag.

① Carry Flag (Cy)

② Parity Flag (P)

③ Auxiliary Carry Flag (AC)

④ Zero Flag (Z)

⑤ Sign Flag (S)

① Carry Flag (CY)

- After the execution of an arithmetic instruction if a carry is produced, the carry flag Cy is set to 1, otherwise it is 0.
- After the addition of two 8-bit numbers, if the sum is larger than 8-bits, a carry is produced & the carry flag is set to 1.

② Parity Flag (P)

- The parity flag P is set to 1, if the result of an arithmetic or logical operation contains even number of 1's. It is reset on 0, if the result contains odd number of 1's.

③ Auxiliary Carry Flag (AC)

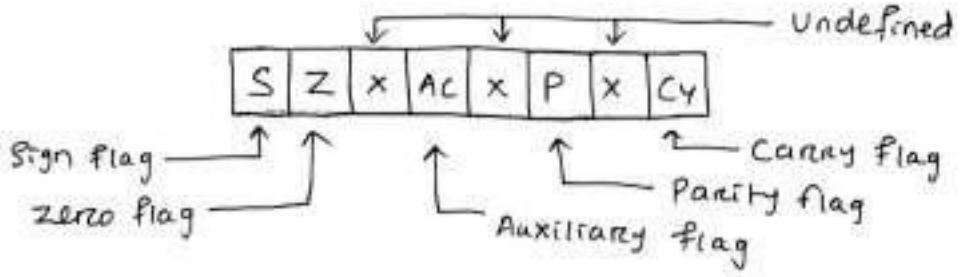
- The auxiliary carry flag holds carry out of the bit number 4 to the bit number 5 resulting from the execution of an arithmetic operation.

④ Zero Flag (Z)

The zero status flag Z is set to 1, if the result of an arithmetic or logical operation is 0. If the result is not zero, the flag is set to 0.

⑤ Sign Flag (S)

The sign flag S is set to 1, if the result of an arithmetic or logical operation is 0. If the result is positive, the sign flag is set to 0.

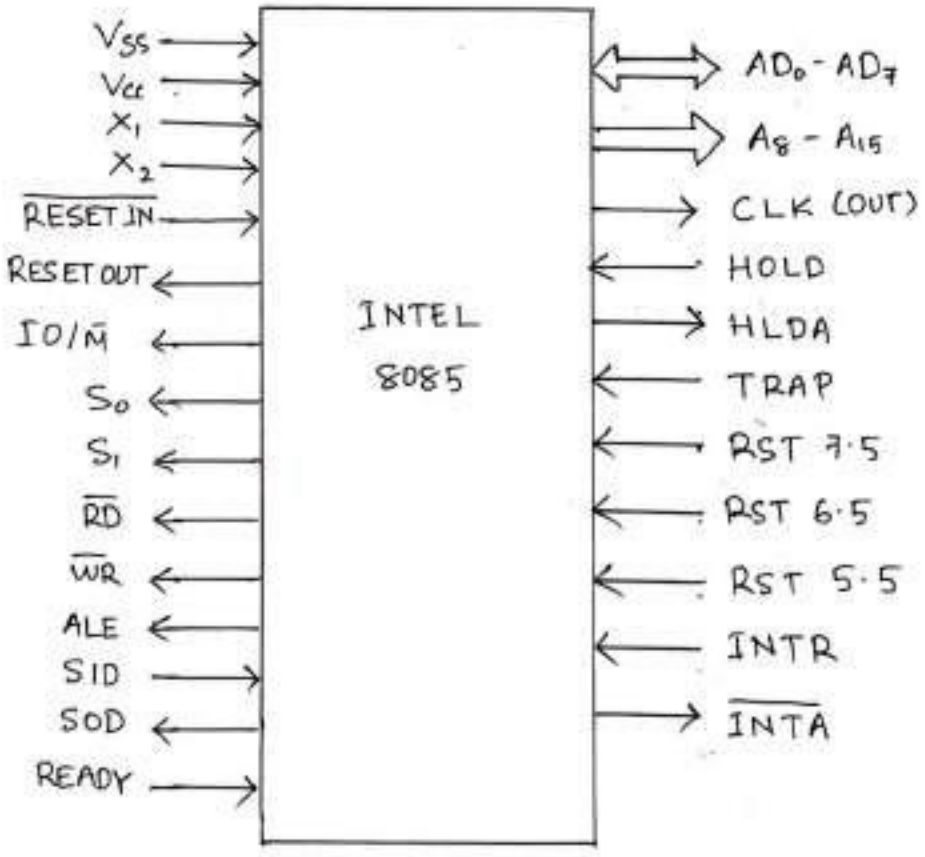


ex ADD CB & E9

```
      (1)  (1)
CB →  1 1 0 0 1 0 1 1
E9 →  1 1 1 0 1 0 0 1
      -----------------
      1 0 1 1 0 1 0 0
```

- Result is non zero ; Z = 0
- MSB of the SUM is 1 ; S = 1 i.e result is negative.
- There is a carry from 4th bit to 5th : AC = 1
- There is a carry generated at the final bit or MSB ; CY = 1
- There are 4 no. of 1s ; P = 1 (even no. of 1)

Pin Diagram



A8 - A15

These are address bus & are used for the most significant bits of the memory address or 8-bits of I/O address.

$AD_0 - AD_7$

These are time multiplexed address/data bus i.e they serve dual purpose. They are used for the least significant 8-bits of the memory address or I/O address during the first clock cycle of a machine cycle. Again they are used for data during second & third clock cycles.

ALE (Address Latch Enable)

- It is an address latch enable signal. It goes high during first clock cycle of a machine cycle & enables the lower 8-bits of the address to be latched. It goes low for data operation.

$IO/\overline{M}$

- It is a status signal which distinguishes whether the address is for memory or I/O.
- When it goes high, the address on the address bus is for an I/O device. When it goes low, the address on the address bus is for a memory location.

S_0, S_1

These are status signals sent by the microprocessor to distinguish the various types of operation.

S_1	S_0	operation
0	0	Halt
0	1	write
1	0	Read
1	1	Fetch

$\overline{RD}$

When microprocessor reads data from a memory location or input device, it is called Read operation. $\overline{RD}$ is a signal sent by the microprocessor to the memory / input device to control Read operation when it goes low, the selected memory or input device is read.

$\overline{WR}$

When microprocessor sends data to a memory location or an output device, it is called write operation. $\overline{WR}$ is a signal sent by the microprocessor to the memory/output device to control write operation. When it goes low, the data which is on the data bus, is written into the selected memory or sent to the output device.

READY

- It is a signal sent by an input or output device to the microprocessor This signal indicates that the input or output device is ready to send or receive data.
- The microprocessor examines READY signal before it performs data transfer operation. A slow input or output device is connected to the microprocessor through READY line.
- When READY is high, it indicates that the input or output divice is ready to send or receive data. When READY is low, the microprocessor waits till READY becomes high. The microprocessor examines the status of READY signal in the second clock cycle of the machine cycle.

HOLD

- When another divice of the computer system, requires address & data buses for data transfer, it sends HOLD signal to the microprocessor. After receiving the HOLD request, the microprocessor sends out a HLDA (HOLD Acknowledge) signal to the device.
- Then the microprocessor leaves the control over the buses as soon as the current machine cycle is completed. The microprocessor regains the control over the buses after the HOLD signal is removed.

HLDA

- It is a HOLD acknowledge signal sent out by the microprocessor after receiving the HOLD signal. It is sent to the device which has issued the HOLD signal. After the removal of the HOLD signal, the HLDA goes low, and thereafter the microprocessor takes over the buses.

INTR

- It is an interrupt signal sent by an external device to the microprocessor. Through this line an external device informs microprocessor that it is ready to transfer data or to initiate certain operation.
- The 8085 microprocessor has 5 interrupt lines. The INTR is one of them. When it goes high, the microprocessor suspends the execution of its normal sequence of instructions after completing the current instruction at hand, it attends the interrupting device.

$\overline{\text{INTA}}$

It is an interrupt acknowledge signal issued by the microprocessor after receiving an interrupt request from an external device. It is a low active signal.

RST 5.5, 6.5, 7.5 & TRAP

These are interrupts. When an interrupt is recognised the next instruction is executed from a fixed location in the memory.

TRAP — 0024
RST 5.5 — 002C
RST 6.5 — 0034
RST 7.5 — 003C

- The TRAP has the highest priority among interrupts. It is a nonmaskable interrupt.

RESET IN

It resets the program counter to zero. It also resets interrupt enable & HLDA flipflops. It does not affect any other flag or register except the instruction register. The CPU is held in reset condition as long as RESET is applied.

RESET OUT

It indicates that the CPU is being reset.

X_1, X_2

These are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

CLK

It is a clock output for user, which can be used for other digital ICs. Its frequency is same at which processor operates.

SID

It is data line for serial input. This pin is used for receiving the data into microprocessor serially.

SOD

It is a data line for serial output. This pin is used for sending the data from the microprocessor serially.

V_{cc}

+5 volts supply

V_{ss}

Ground reference

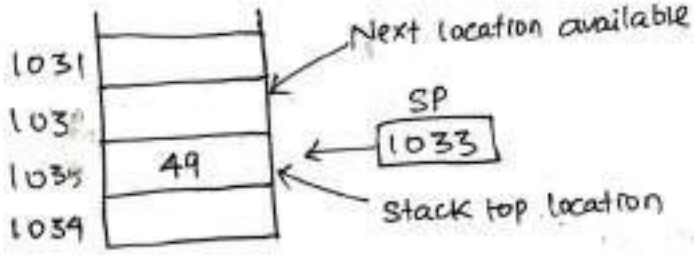
STACK, STACK POINTER, STACK TOP

During the execution of a program sometimes it becomes necessary to save the contents of certain registers because the registers are required for some other operation.

- These contents are moved to certain memory locations by PUSH operation. After completing the operations those contents which were saved in the memory are transferred back to the registers by POP operation.
- The memory locations kept for this purpose is called stack.

Stack top :- The last memory location of the occupied portion of the stack is called stacktop.

Stack pointer :- A special purpose 16-bit register known as stack pointer holds the address of stacktop.



- Data are stored in the stack on last-in-first-out (LIFO) principle.
- Push - To add an element to the stack.
- Pop - To remove an element from the stack.

Interrupt

When microprocessor receives any interrupt signal from peripherals which are requesting its services, it stops its current execution & program control is transferred to a sub-routine by generating CALL signal & after executing sub-routine by generating RET signal again program control is transferred to main program from where it had stopped.

Hardware & Software interrupts

- When microprocessors receive interrupt signals through pins of microprocessor, they are known as Hardware interrupts.
- There are 5 Hardware interrupts in 8085 microprocessor. They are INTA, RST 7.5, RST 6.5, RST 5.5, TRAP.
- Software interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor.
- There are 8 Software interrupts in 8085 microprocessor. They are RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

Vectored & Non-vectored interrupts

- Vectored interrupts are those which have fixed vector address & after executing these, program control is transferred to that address.
- Non-vectored interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts.
- INTR is the only non-vectored interrupt is 8085 microprocessor.

Maskable & Non maskable Interrupts

Maskable interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled.

- INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor.
- Non maskable interrupts are those which can not be disable or ignored by microprocessor.
- TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering.

Priority of Interrupts

TRAP	Highest
RST 7.5	↓
RST 6.5	
RST 5.5	
INTR	Lowest

OPcode & Operand.

Each instruction contains two parts: operation code (opcode) & operand.

Opcode: The first part of an instruction which specifies the task to be performed by the computer is called opcode.

Operand: The second part of the instruction is the data to be operated on, and it is called operand.

- The operand given in the instruction may be in various forms such as 8-bit or 16-bit data, 8-bit or 16-bit address, internal registeres or a register or memory location.
- In some instructions the operand is implicit.
- When operand is a register it is understood that the data is the content of the register.

Instruction word size

According to the word size the Intel 8085 instructions are classified into 3 types:

① 1-Byte instruction
② 2-Byte instruction
③ 3-Byte instruction

① one-Byte instruction

A 1-Byte instruction includes the opcode & the operand in the same Byte.

ex
```
MOV A, B
```
- Copy the content of the B register in Accumulator.

```
ADD B
```
- Add the content of register B to the content of the accumulator.

```
CMA
```
- Invert or complement each bit in the accumulator.

② Two Byte instruction

- In a two-byte instruction the 1st byte of the instruction is its opcode & the 2nd byte is either data or address.

ex

MVI B, 05

move data 05 to register B

IN 01

Accept data byte from an input device & place it in the accumulator.

③ Three Byte instruction

- In a three byte instruction the 1st byte of the instruction is its opcode & the 2nd & 3rd bytes are either 16-bit data or 16-bit address.

ex

LXI H, 2400H

Load H-L pair with 2400H

LDA 2500H

Get the Content of the memory location 2500H into accumulator.

Instruction Set of 8085

Data Transfer

Mov r_1, r_2

- move content of r_2 register to r_1 register.

ex = MOV A, B [content of register B moves to register A]

- 1 Byte instruction.

Mov r, M

- Move content of memory to register r.

$[r] \leftarrow [[H\text{-}L]]$

- The content of the memory location, whose address is in H-L pair, is moved to register r.

- 1-Byte instruction.

Mov M, r

- move the content of register to memory.

$[[H\text{-}L]] \leftarrow [r]$

ex = Mov M, C [moves content of register C to the memory location whose address is in H-L pair]

MVI r, data

- Move immediate data to register.

- It is a 2-Byte instruction. 1st Byte is Opcode, 2nd Byte is data.

ex = MVI A, 15 [data 15 moves to register A]

MVI M, data

- move immediate data to memory
- 2 Byte instruction.

MVI M, 50 [move 50 to memory location whose address is in HL pair]

LXI rp, data 16 bit

- Load 16-bit immediate data into register pair rp.
- If LXI H is mentioned, it denotes HL pair.

ex

LXI H, 2050H [It loads 2050 into HL pair

- It is a 3-Byte instruction. 1st Byte is LXI H, 2nd Byte is 20 & 3rd Byte is 50.

LDA address

The content of the memory location specified in the instruction is loaded into the accumulator.

ex LDA 3020H [content of 3020 moves to Accumulator]

- It is a 3 Byte instruction

STA address

The content of the accumulator is stored in the memory location specified in the instruction.

ex STA 6523H [Content of accumulator Store in 6523H]

- It is a 3 Byte instruction.

LDAX rp

The content of the memory location, whose address is in the register pair rp, is loaded into the accumulator.

ex LDAX B [load the content of the memory location, whose address is in the B-C pair into the accumulator]

- It is a 1-Byte instruction.

STAX rp

The content of the accumulator is stored in the memory location whose memory address is in the register pair rp.

ex STAX D [store the content of the accumulator in the memory location whose address is in DE pair]

- It is a 1-Byte instruction.

Arithmetic Group

ADD r

- The content of register r is added to the content of the accumulator & the sum is placed in the accumulator.

ADD M

The content of memory location addressed by H-L pair is added to the content of the accumulator.

ADI data

The immediate data is added to the content of the accumulator.

ex ADI, 55H

- 2 Byte instruction

SUB r

- The content of register r is subtracted from the content of the accumulator.
- The result is placed in the accumulator.

SUI data

- The immediate data is subtracted from the content of the accumulator. The result is placed in the accumulator.

INR r

- The content of register r is incremented by one.

$[r] \leftarrow [r] + 1$

INR M

The content of the memory location addressed by HL pair is incremented by one.

$[[H\text{-}L]] \leftarrow [[H\text{-}L]] + 1$

DCR r

The content of register r is decremented by one.

DCR M

The content of the memory location addressed by H-L pair is decremented by one.

INX rp

The content of the register pair rp is incremented by one.

DCX rp

The content of the register pair rp is decremented by one.

Logical Group

ANA R

The content of register R is ANDed with the content of the accumulator & result is placed in the accumulator.

ANI data

The data is ANDed with the content of the accumulator.
- 2 Byte instruction.

ORA R

The content of register R is ORed with the content of the accumulator.

ORI data

The data in the instruction is ORed with the content of the accumulator.

XRA R

The content of register R is XORed with the content of the accumulator.

CMC

The carry flag is complemented.

RLC

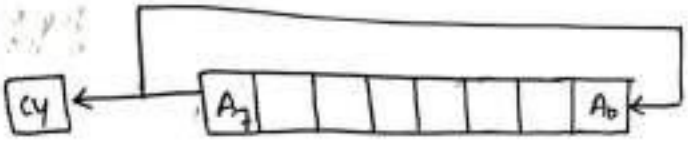
The content of the accumulator is rotated left by one bit

RRC

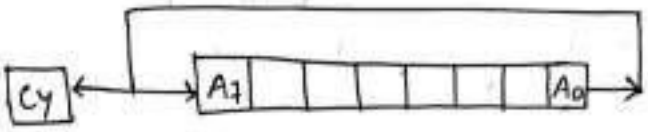
The content of the accumulator is rotated right by one bit.

RAR

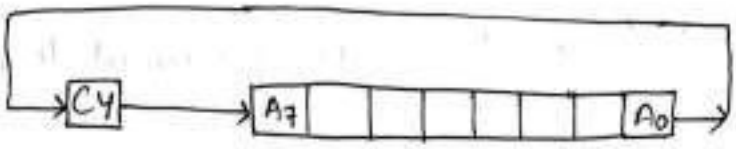
The content of the accumulator is rotated right one bit through carry



(RLC)



(RRC)



(RAR)

Branch Group

JNZ address

The program jumps to the instruction specified by the address, if the result is non-zero

JC address

The program jumps to the instruction specified by the address if there is a carry.

JNC address

The program jumps to the instruction specified by the address if there is no carry.

JZ address

The program jumps to the instruction specified by the address if the result is zero.

Addressing Mode

There are various techniques to specify data for instructions. These techniques are called addressing modes.

1. Direct addressing mode
2. Register addressing mode
3. Register indirect addressing mode
4. Immediate addressing mode
5. Implicit addressing mode

1. Direct Addressing mode

- In this mode of addressing the address of the operand is given in the instruction itself.

ex STA 1000H

[Store the content of the accumulator in the memory location 1000H]

2. Register Addressing mode

In register addressing mode the operand is in one of the general purpose registers

ex MOV A, B

[move the content of register B to register A]

3. Register indirect addressing mode

In this mode of addressing the address of the operand is specified by a register pair.

ex MOV A, M

move the content of the memory location, whose address is in HL pair to the accumulator.

4. Immediate Addressing mode

In immediate addressing mode the operand is specified within the instruction itself.

ex

```
MVI A, A5      [move A5 to register A]
ADI 10         [Add 10 to the content of the accumulator]
```

5. Implicit Addressing mode (Implied Addressing mode)

These instructions do not require the address or data. It is implied in the instruction itself.

```
CMC      [Complement the carry flag]
RAL      [Rotate accumulator left through carry]
```

Instruction Cycle

This is the time required by the microprocessor to fetch & execute one complete instruction.

The instruction cycle is in two parts :-

1. Fetch cycle
2. Execute Cycle

Fetch cycle

- This is the time required by the microprocessor to fetch all bytes of an instruction.
- The length of the fetch cycle is thus determined by the no. of bytes in an instruction.

Execution cycle

This is the time required by the microprocessor to execute a fetched instruction.

T-State

A T-state is one clock cycle of the microprocessor.

T = clock period = 1/clock frequency

Machine cycle

It is the time required by the microprocessor doing one operation & accessing one byte from the external module. (memory or I/O)

Timing Diagram

Opcode fetch

- This cycle is used to fetch the opcode from the memory.
- This is the 1st machine cycle of every instruction.
- It is generally of 4T states but for some instruction it is 6T.

During T_1

- $A_{15}-A_8$ contains the higher byte of the address (PCH)
- As ALE is high AD_7-AD_0 contains the lower byte of the address (PCL)
- Since it is an opcode fetch cycle, S_1 & S_0 go high.
- Since it is a memory operation $IO/\overline{M}$ goes low.

During T_2

- As ALE goes low address is removed from AD_7-AD_0.
- As RD goes low, data appears on AD_7-AD_0

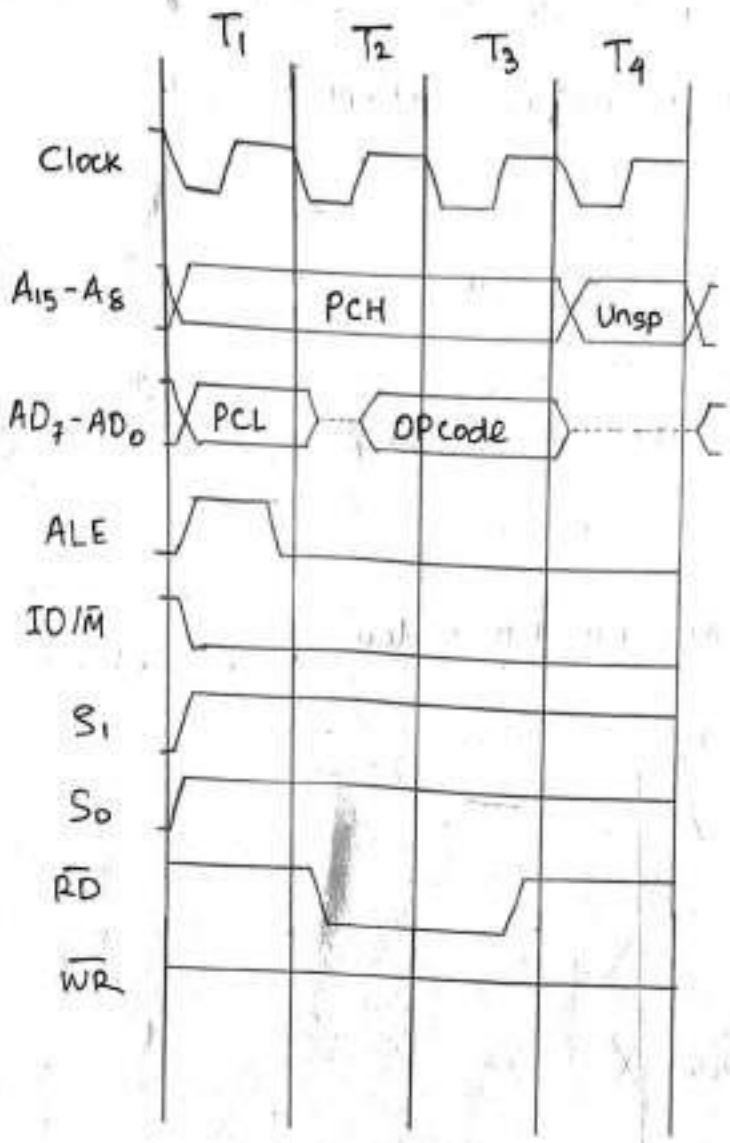
During T_3

- Data remains on AD_7-AD_0 till RD is low.

During T_4

- T_4 State is used by the microprocessor to decode the opcode.

Operation	$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$	S_1	S_0	T-state
opcode fetch	0	0	1	1	1	4/6
Memory Read	0	0	1	1	0	3
Memory write	0	1	0	0	1	3
IO Read	1	0	1	1	0	3
IO write	1	1	0	0	1	3



Memory Read

- This cycle is used to fetch one byte from the memory.
- This cycle can be used to fetch the operand bytes of an instruction or any data from the memory
- It requires 3T states.

During T1

- $A_{15} - A_8$ contains the higher byte of the address (PCH)
- As ALE is high, $AD_7 - AD_0$ contains the lower byte of the address (PCL).

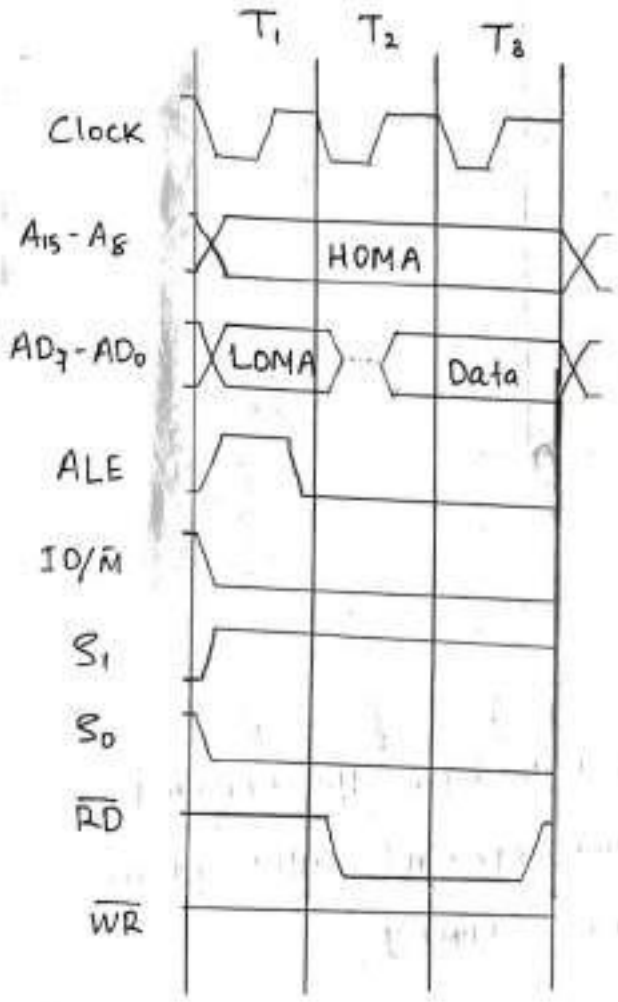
- Since it is a memory Read cycle, S_1 goes high & S_0 goes low.
- Since it is a memory operation, IO/$\bar{M}$ goes low.

During T_2

- ALE goes low
- Address is removed from $AD_7 - AD_0$
- As RD goes low, data appears on $AD_7 - AD_0$.

During T_3

- Data remains on $AD_7 - AD_0$ till RD is low



Memory write

- This cycle is used to send one byte into the memory.
- It requires 3 T-states.

During T_1

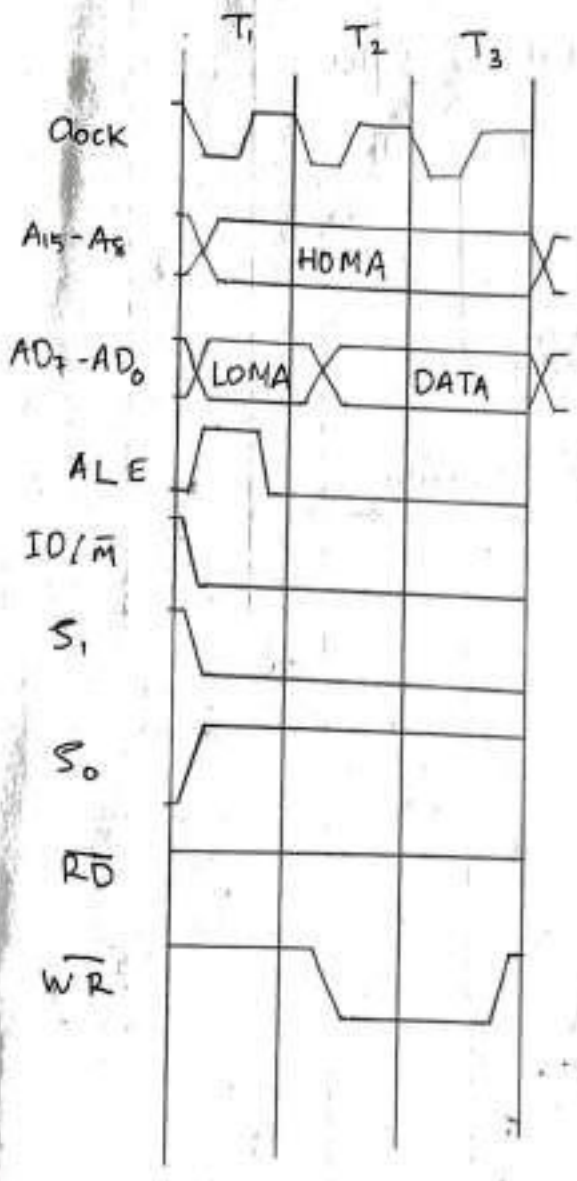
- A_{15} - A_8 contains the higher byte of the address (PCH)
- As ALE is high. AD_7 - AD_0 contains the lower byte of the address (PCL)
- Since it is a memory write operation, S_0 goes high & S_1 goes low.
- Since it is a memory operation IO/$\bar{M}$ goes low.

During T_2

- ALE goes low
- Address is removed from AD_7 - AD_0.
- Data appears on AD_7 - AD_0 & WR goes low.

During T_3

- Data remains on AD_7 - AD_0 till WR is low.



IO Read

- This cycle is used to fetch one byte from an IO port.
- It requires 3 T-states.

During T_1

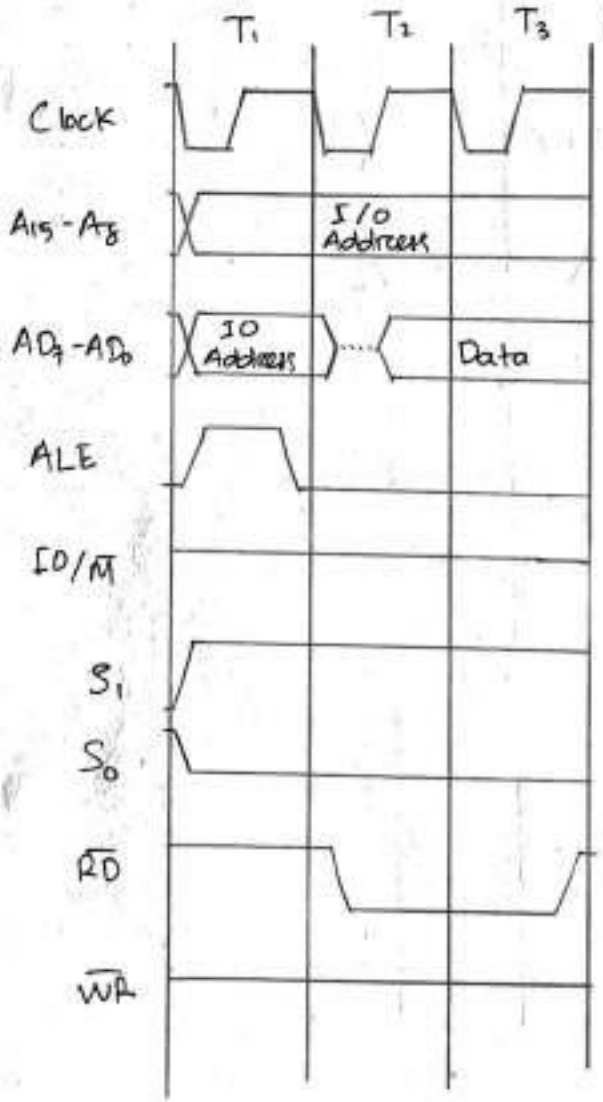
- The lower 8 bits of the IO port address are duplicated into the higher order address bus $A_{15}-A_8$.
- AS ALE is high AD_7-AD_0 contains the lower byte of the address
- Since it is a read operation S_1 goes high & S_0 goes low.
- Since it is an IO operation $IO/\overline{M}$ goes high.

During T_2

- ALE goes low.
- Address is removed from AD_7-AD_0.
- As RD goes low, data appears on AD_7-AD_0.

During T_3

- Data remains on AD_7-AD_0 till RD is low.



IO Write

- This cycle is used to send one byte into an IO port.
- It requires 3-T states.

During T_1

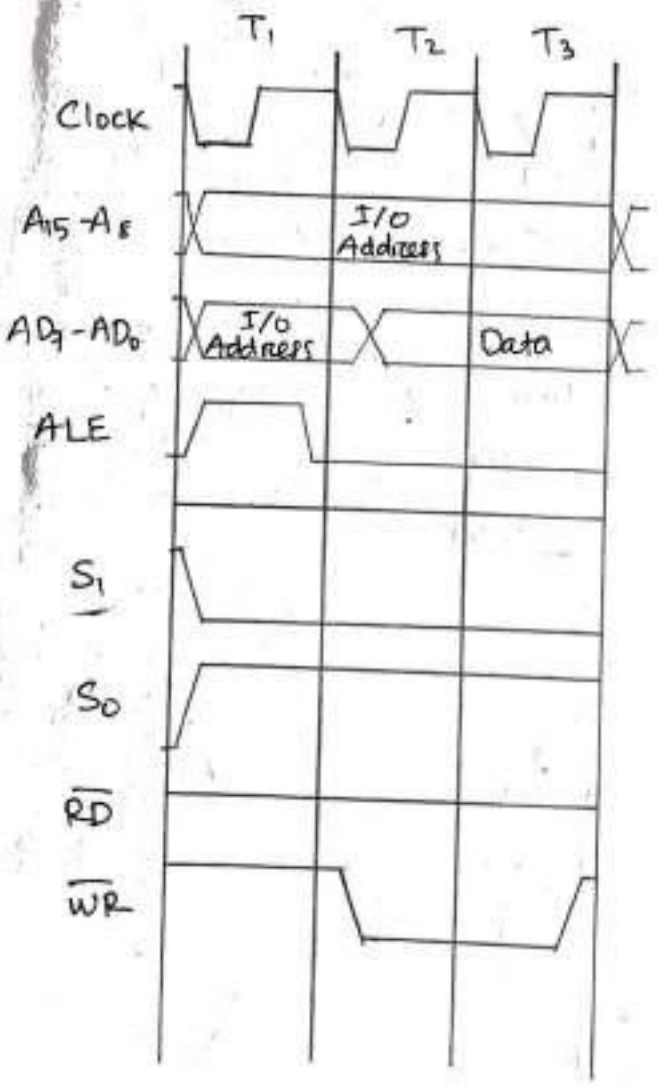
- The lower 8 bits of the IO port address are duplicated into the higher order address bus $A_{15}-A_8$.
- As ALE is high AD_7-AD_0 contains the lower byte of the address
- Since it is an IO write cycle, S_0 goes high & S_1 goes low.
- Since it is an IO operation, IO/$\overline{M}$ goes high.

During T_2

- ALE goes low
- Address is removed from AD_7-AD_0
- Data appears on AD_7-AD_0 & WR goes low.

During T_3

- Data remains on AD_7-AD_0 till RD is low.



Timing Diagram for 8085 instructions

MVI B, 25H

opcode fetch → 4 T-States
Memory Read → 3 T-states

LXI B, 2000H

opcode fetch → 4 T-states
Memory Read → 3 T-States
Memory Read → 3 T-States

LDA 2000H

opcode fetch → 4 T-States
Memory Read → 3 T-States
Memory Read → 3 T-States
Memory Read → 3 T-States

MOV B, C

opcode fetch → 4 T-States

INR M

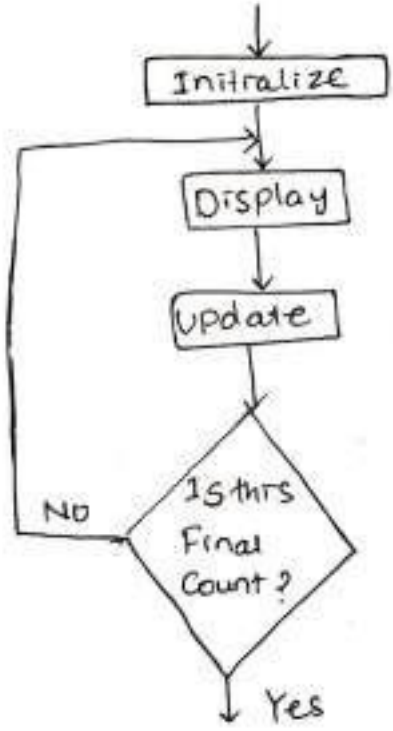
opcode fetch → 4 T-States
Memory Read → 3 T-States
memory write → 3 T-States

OUT 80H

opcode fetch → 4 T-States
memory Read → 3 T-States
I/O Write → 3 T-States

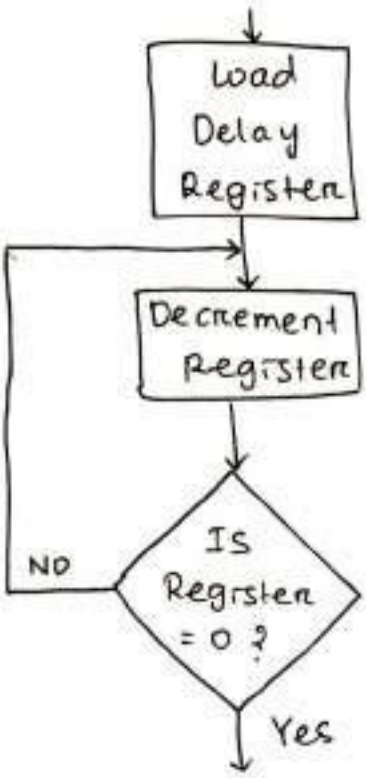
Counter

- A counter is designed simply by loading an appropriate number into one of the registers & using the INR (Increment by one) or the DCR (Decrement by one) instructions.
- A loop is established to update the count.
- Each count is checked to determine whether it has reached the final number; if not, the loop is repeated.



Time Delay

- The procedure used to design a specific delay is similar to that used to set up a counter.
- A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero by setting up a loop with a conditional jump instruction.
- The loop causes the delay, depending upon the clock period of the system.



Time Delay Using one Register

- A count is loaded in a register, and we can use a loop to produce a certain amount of time delay in a program.

Delay:	MVI B, 8 bit count	7 T-States
Loop:	DCR B	4 T-States
	JNZ Loop	10/7 T-States

Total no. of T-States = T-states outside the loop + T-States of the loop

- Conditional jump instruction needs 10 T-states when it jumps or true & it needs 7 T-states when false or exit the loop.

- If clock frequency is 2MHz, clock time period $T = \frac{1}{2\,MHz} = 0.5\,\mu sec$

  one T-State = 0.5 μsec

- Time required to execute MVI instruction : 7 T-state

$= 7 \times 0.5$ μsec

$= 3.5$ μsec

- Max^m count can be FFH i.e $(255)_{10}$.

- Time delay in the loop is $(4T + 10T) \times (255-1) + (4T+7T)$

(It is true for 254 times & false once)

$14T \times 254 + 11T = 14 \times 0.5 \times 254 + 11 \times 0.5$

$= 1778 + 5.5 = 1783.5$ μsec

∴ Total time delay $= 3.5 + 1778 + 5.5 = 1787$ μsec

Time Delay using a Register pair

```
Delay : LXI B , 16 bit Count  ; Load Bc with 16 bit count   ; 10 T
Loop  : DCX B                 ; Decrement Bc by one         ; 6 T
        MOV A , C             ; place contents of C in A    ; 4 T
        ORA B                 ; OR B with C                 ; 4 T
        JNZ LOOP              ; If result ≠0, jump back to loop ; 10/7 T
```

- Time delay of LXI instruction : $10T = 10 \times 0.5 = 5$ μsec

- Maximum count can be FFFFH i.e $(65535)_{10}$.

- Time delay of the loop : $(6+4+4+10)T \times 65534 + (6+4+4+7)T$

$= 24T \times 65534 + 21T$

$= 24 \times 0.5 \times 65534 + 21 \times 0.5$

$= 786408 + 10.5 = 786418.5$ μsec

∴ Total Time Delay $= 5 + 786418.5 = 786423.5$ μsec $= 786.423$ ms

Time Delay using a Loop within a Loop

```
        MVI  B, count1  ; move count1 to B                    ; 7T
Loop2:  MVI  C, count2  ; move count2 to C                    ; 7T
Loop1;  DCR  C          ; Decrement Content of C by 1         ; 4T
        JNZ  Loop1      ; If result ≠0; jump back to loop1    ; 10/7 T
        DCR  B          ; Decrement content of B by 1         ; 4T
        JNZ  Loop2      ; If result ≠0; jump back to Loop2    ; 10/7 T
```

- Time Delay of MVI B instruction : $7T = 7 \times 0.5 = 3.5\ \mu sec$
- Maximum Count can be FF. So $Count1_{max} = Count2_{max} = FF$
- Time Delay of Loop1 : $(10+4)T \times 254 + (4+7)T$

$$= 14 \times 0.5 \times 254 + 11 \times 0.5$$
$$= 1783.5\ \mu sec$$

- Time Delay of Loop2 : $255\,[T_{loop1} + 21T] - 3T$

$$= 255\,[1783.5 + 21 \times 0.5] - 3 \times 0.5$$
$$= 255 \times 1794 - 1.5$$
$$= 457471.5\ \mu sec$$

- Total Time Delay = Time of Loop2 + 3.5 μsec

$$= 457471.5 + 3.5$$
$$= 457475\ \mu sec$$
$$= 457.475\ msec$$

Note

Using NOP instruction :-

4 T-state ⇒ $4 \times 0.5\ \mu sec = 2\ \mu sec$

Assembly Language Programming of 8085

ex1 : Place 5C in register B.

```
MVI  B, 5C      [5C is moved to register B]
HLT             [stop ]
```

ex2 : Place 05 in the accumulator. Increment it by one & store the result in the location 2050 H

Program

```
MVI  A, 05      [05 is moved to A , A ← 05 ]
INR  A          [Increment the content of A by 1 ]
STA  2050 H     [Store content of A in 2050H ]
HLT             [ Halt /stop ]
```

ex3 : Addition of two 8-bit numbers , Sum 8 bit

Add 49H & 56H

The 1st number 49H is in the memory location 2501 H

The 2nd number 56H is in the memory location 2502 H

The result is to be stored in the memory location 2503H

Program

```
LXI  H, 2051 H      [ H-L pair ← 2051 ]
MOV  A, M           [ move content of M to A ]
INX  H              [Increment content of H-L pair]
ADD  M              [Add content of M & A ]
STA  2503 H         [ Store content of A in 2503H]
HLT                 [Hall (Stop)]
```

Ex4 : Write a program or ALP to subtract two 8-bit number. 1st number is in 2501H, 2nd number is in 2502H & Store the result in 2503H.

Program

```
LXI  H, 2501 H    [ H-L ← 2501 ]
MOV  A, M         [ Move content of M to A ]
INX  H            [ H-L pair incremented by 1 ]
SUB  M            [ Subtract content of M from A ]
INX  H            [ Increment HL pair by 1 ]
MOV  M, A         [ Store content of A in M ]
HLT               [ Halt / Stop ]
```

Ex5 : Write a program to add two 8-bit numbers. Sum is 16 bit.
1st number → 2501, 2nd number → 2502, result → 2503, 2504

Program

```
      LXI  H, 2501 H    [ H-L ← 2501 ]
      MVI  C, 00        [ Move 00 into C ]
      MOV  A, M         [ Move content of M to A ]
      INX  H            [ H-L pair incremented by one ]
      ADD  M            [ Add content of A & M ]
      JNC  AHEAD        [ If no carry then go to AHEAD ]
      INR  C            [ Increment content of 'C' by one ]
AHEAD STA  2503 H       [ Store content of A in 2503 ]
      MOV  A, C         [ Move content of C to A ]
      STA  2504 H       [ Store content of A to 2504 ]
      HLT               [ Halt ]
```

ex 6: Write an Assembly language program to compare two 8 bit data stored in 3001 & 3002. Store the smaller number in 3003.

Program

```
        LXI  H  3001     [HL ← 3001]
        mov  A, M        [move content of M to A]
        INX  H           [Increment H-L pair by 1]
        CMP  M           [A-M]
        JC   SKIP        [Jump if carry = 1]
        MOV  A, M        [move content of M to A]
SKIP:   INX  H           [Increment H-L pair by 1]
        MOV  M, A        [move content of A to M]
        HLT              [Stop / Halt]
```

ex 7: Decimal addition of two 8-bit numbers, sum is 16 bit.

Program

```
        LXI  H, 2501H    [H-L ← 2501H]
        MVI  C, 00       [C ← 00]
        MOV  A, M        [A ← M, [HL]]
        INX  H           [H-L ← H-L+1 i.e 2502H]
        ADD  M           [Add content of A & M]
        DAA              [Decimal Adjust Accumulator]
        JNC  AHEAD       [If carry = 0, go to AHEAD]
        INR  C           [Increment C]
AHEAD:  STA  2503H       [Store accumulan into 2503H]
        MOV  A, C        [move C to A]
        STA  2504H       [Store A into 2504H]
        HLT              [Halt]
```

ex 8 : Find one's complement of an 8-bit number.

Program

```
LDA   2501H     [Store the content of 2501 in A]
CMA             [complement the content of A]
STA   2502H     [Store the content of A in 2502]
HLT             [Halt]
```

ex 9 : Find two's complement of an 8-bit number.

Program

```
LDA   2501H     [Load A with content of 2501]
CMA             [Complement the content of A]
INR   A         [Increment the content of A]
STA   2502H     [Store the content of A in 2502]
HLT             [Halt]
```

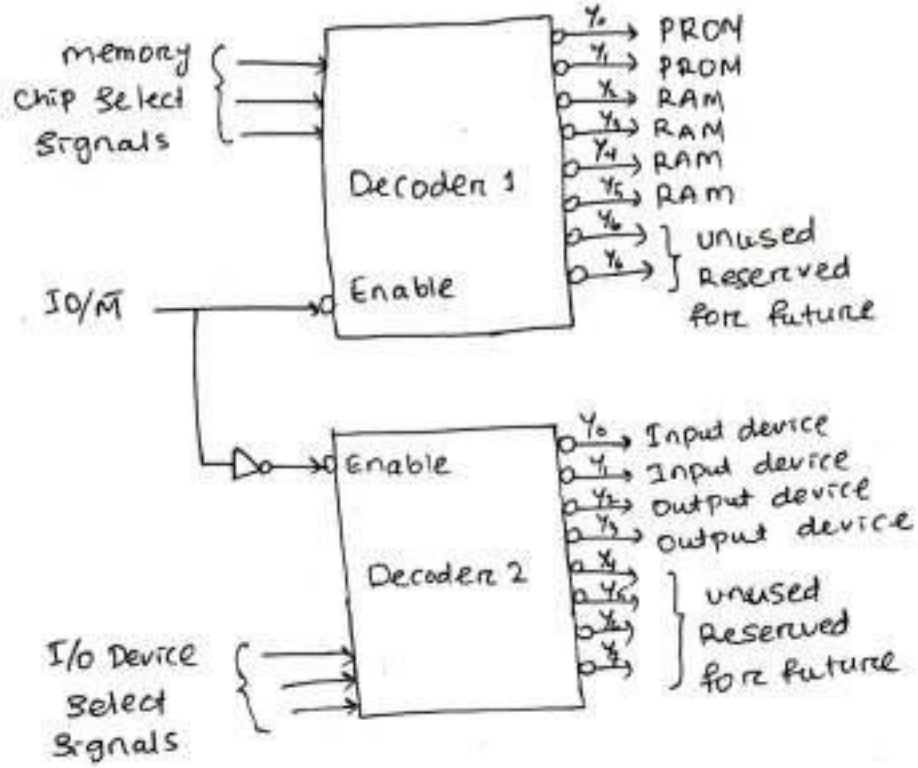
ex 10 : write a program to find the largest number in a data Array.

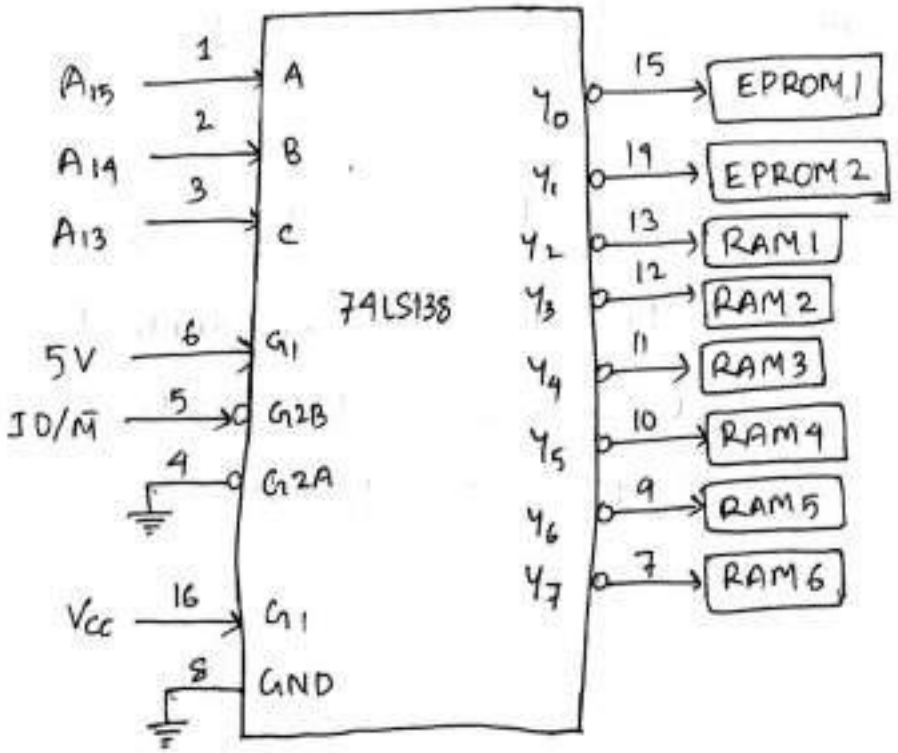
Program

```
      LXI  H, 2500H     [H-L pair ← 2500]
      MOV  C, M   (Count) [Move the content of M in C]
      INX  H            [Increment H-L pair]
      MOV  A, M         [move content of M to A]
      DCR  C            [Decrement the content of C]
LOOP: INX  H            [Increment HL pair]
      CMP  M            [compair M with A]
      JNC  AHEAD        [Jump AHEAD if no carry]
      MOV  A, M         [Move the content of M to A]
AHEAD: DCR C            [Decrement the content of C]
      JNZ  LOOP         [Jump LOOP if result (not 0]
      STA  2450         [Store content of A in 2450]
      HLT               [Halt]
```

Memory Interfacing

The address of a memory location or an I/O device is sent out by the microprocessor. The corresponding memory chip or I/O device is selected by a decoding circuit.





G_1, G_{2A}, G_{2B} are enable signals.

- To enable the chip, G_1 should be high, and G_{2A} and G_{2B} should be low. A, B and C are select lines.

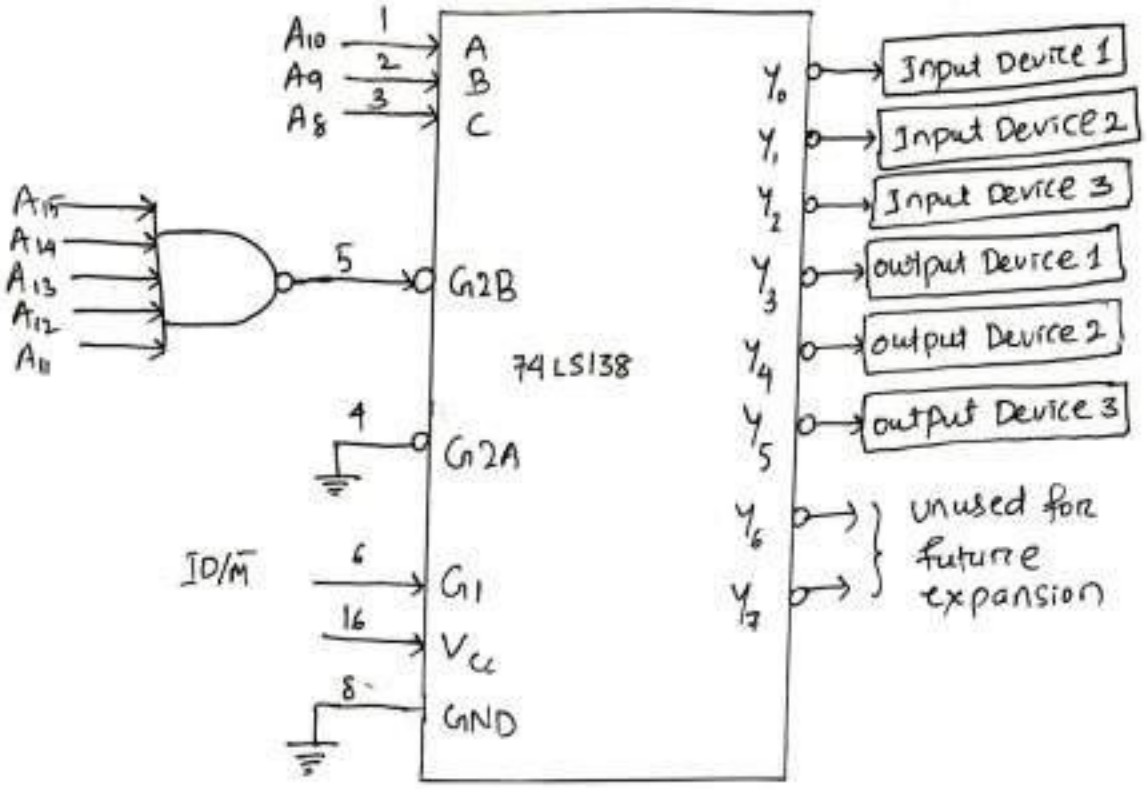
- By applying proper logic to select lines any one of the outputs can be selected.

- $Y_0, Y_1 \ldots Y_7$ are 8 output lines. An output lines goes low when it is selected. other output lines remain high.

Decoder output	Memory device	memory location address
Y_0	EPROM 1	0000 to 1FFF
Y_1	EPROM 2	2000 to 3FFF
Y_2	RAM 1	4000 to 5FFF
Y_3	RAM 2	6000 to 7FFF
Y_4	RAM 3	8000 to 9FFF
Y_5	RAM 4	A000 to BFFF
Y_6	RAM 5	C000 to DFFF
Y_7	RAM 6	E000 to FFFF

- The entire memory address has been divided into 8 zones.
- Address lines A_{15}, A_{14} & A_{13} have been applied to the select lines A, B and C. other address lines $A_0, A_1, A_2, \ldots$ and A_{12} go directly to memory chip

I/O Interfacing

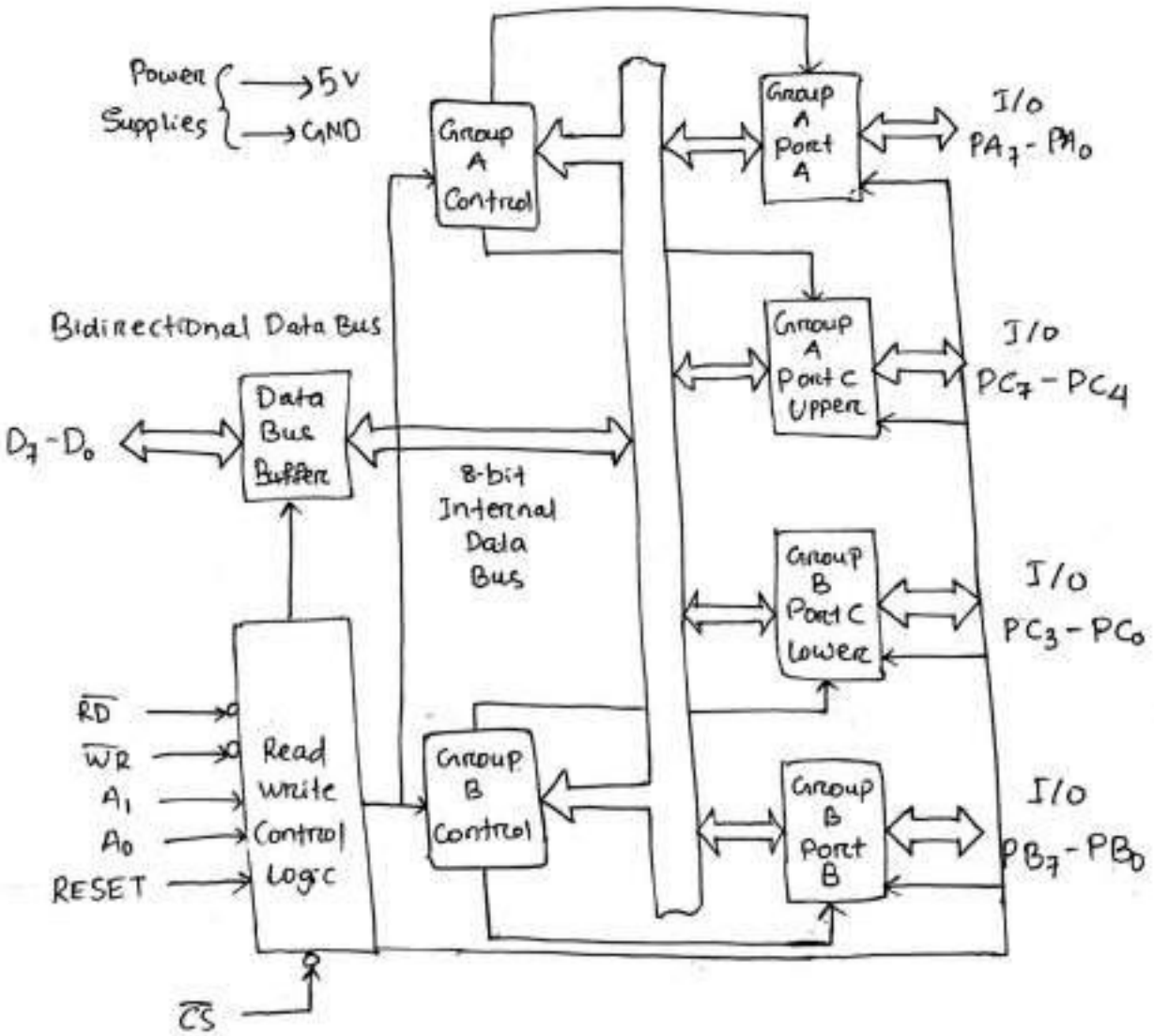


- As the address of an I/O device is of 8 bits, only A_{15}-A_8 lines of address bus are used for I/O addressing.
- The address lines A_8, A_9 & A_{10} have been applied to select lines A, B & C of the chip. The address lines A_{11} - A_{15} are applied to G2B through a NAND gate.
- G2B becomes low only when all address lines A_{11}-A_{15} are 1.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	selected o/p lines	corresponding Address	I/O Device
1	1	1	1	1	0	0	0	Y_0	F8	Input Device 1
1	1	1	1	1	0	0	1	Y_1	F9	Input Device 2
1	1	1	1	1	0	1	0	Y_2	FA	Input Device 3
1	1	1	1	1	0	1	1	Y_3	FB	output Device 1
1	1	1	1	1	1	0	0	Y_4	FC	output Device 2
1	1	1	1	1	1	0	1	Y_5	FD	output Device 3
1	1	1	1	1	1	1	0	Y_6	FE	Unused
1	1	1	1	1	1	1	1	Y_7	FF	Unused

Programmable peripheral Interface 8255

- The intel 8255 is a programmable peripheral interface.
- Its main functions are to interface peripheral devices to the microcomputer.
- It has three 8-bit ports, namely port A, port B & port C. The port C has been further devided into two of 4-bit ports, i.e port C upper & port C lower.



- The ports are divided into two groups i.e Group A & Group B.
- Group A has port A & Cupper where as Group B has port B & Clower.
- Each port can be programmed either as an input port or output port.

Chip Select ($\overline{CS}$)

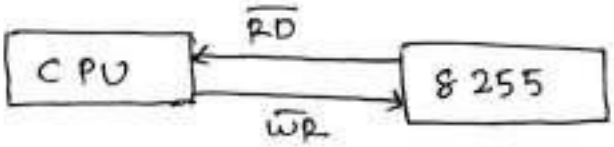
A low on this input selects the chip and enables the communication between the 8255 & the CPU.

$\overline{RD}$ (Read)

when this signal is low 8255 sends out data or status thus information to the CPU on the data bus.

$\overline{WR}$ (write)

A low on this input pin enables the CPU to write data or control words into the 8255.



A_1, A_0

The selection of ports & control word register is done using A_0 & A_1 in conjunction with $\overline{RD}$ & $\overline{WR}$.

Input Operation

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	0	1	0	port A → Data bus
0	1	0	1	0	Port B → Data bus
1	0	0	1	0	port C → Data bus
1	1	0	1	0	control word → Data bus

Output Operation

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	0	Data bus → port A
0	1	1	0	0	Data bus → port B
1	0	1	0	0	Data bus → port C
1	1	1	0	0	Data bus → control word

RESET

A high on this input pin clears the control register & all ports (A, B, C) are initialized to input mode. This is connected to RESET OUT of microprocessor.

Port A, B, C

- These are 8-bit input/output port.
- They have one 8bit data output latch/buffer & one 8-bit input latch.

Group A & Group B control

- The functional configuration of each port is programmed by the system software.
- The control words given by the CPU, configure the associated ports of the each of the two groups.

- The control words contains information like mode set reset etc that initializes the functional configuration of 8255.
- Control word is written into the control register by the CPU of the microprocessor.
- No read operation is associated with it.

Data Bus Buffer

- It is an 8-bit buffer used to interface the chip to the system data bus.
- Data is transmitted or received by the buffer upon execution of IN or OUT instructions by the the CPU.

Read/write Control Logic

- Its function is to control the internal operation of the device and to control the transfer of data & control or status word.
- It accepts inputs from the CPU address & control buses and in turn issues commands to both the control groups.

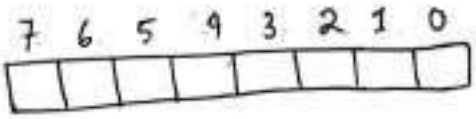
Operating Modes of 8255

8255 has 3 modes of operation :-

(i) Mode 0 - Simple input/output
(ii) Mode 1 - Strobed input/output
(iii) Mode 2 - Bidirectional port.

Control word

Control word Bits →



Bit No. 0 :- It is for port C_{lower}. To make port C_{lower} an input port, the bit is set to 1 & to make output port, bit is set to 0.

Bit No. 1 :- It is for port B. To make port B an input port, the bit is set to 1 & To make output port, the bit is set to 0.

Bit No. 2 :- It is for the selection of the mode for the port B. If the port B has to operate in mode 0, the bit is set to 0. For mode 1 operation of the port B, it is set to 1.

Bit No. 3 :- It is for the port C_{upper}. To make port C_{upper} an input port, the bit is set to 1 & to make output port, bit is set to 0.

Bit No. 4 :- It is for port A. To make port A an input port, the bit is set to 1 & to make port A an output port, bit is set to 0.

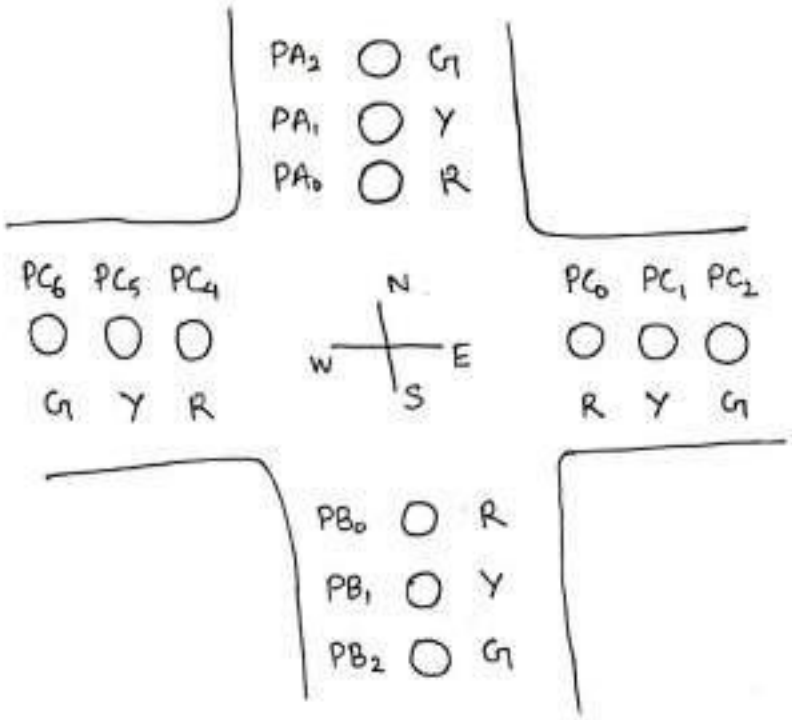
Bit No 5 & 6 :- These bits are to define the operating mode of the port A.

Bit No. 6	Bit No. 5	mode of port A
0	0	mode 0
0	1	mode 1
0	0/1	mode 2

Bit No. 7 :- It is set to 1 if port A, B & C are defined as input/output port. It is set to 0 if the individual pins of the port C are to be set or reset.

Traffic Light Controller

8255 is used to connect between microprocessor & output or input devices.



Port A (PA) → 08

Port B (PB) → 09

Port C (PC) → 0A

Control word register → 0B

- All ports of 8255 have been programmed as output ports because LEDs are output of microprocessor.
- The control word to make all ports output ports in mode 0 operation is 80H (10000000).
- Positive logic has been used to switch on LEDs.

(i) Red light → Does not allow crossing

(ii) Yellow light → To make alert

(iii) Green light → Allow crossing

- Delay I & Delay II are two subroutine used.
- Subroutine is a program which can be used several times in many program & can be called whenever required.

Program

```
        MVI  A,80H
        OUT  0B          [0B ← 80 , select each port as output]
AGAIN:  MVI  A,01H
        OUT  09          [09 for port B , PB ← 01 , Red ON]
        OUT  08          [08 for port A , PA ← 01 , Red ON]
        MVI  A,44H
        OUT  0A          [0A for portC , PC ← 44 , Green ON]
        CALL DELAY 1     [DELAY I for time delay]
        MVI  A, 22H
        OUT  0A          [Port C ← 22, Yellow ON for east,west]
        MVI  A,02H
        OUT  09          [Port B ← 02 , Yellow ON for South]
        OUT  08          [PortA ← 02, Yellow ON for North]
        CALL DELAY II
        MVI  A, 11H
        OUT  0A          [Port C ← 11, Red ON for east,west]
        MVI  A,04H
        OUT  08          [PortA ← 04 , Green ON for North]
        OUT  09          [Port B ← 04 , Green ON for South]
        CALL DELAY I
        MVI  A, 22H
        OUT  0A          [Port C ← 22, Yellow for east,west]
```

```
MVI A, 02H
OUT 09          [Port B ← 02, Yellow ON for South]
OUT 08          [Port A ← 02, Yellow ON for North]
CALL DELAY II
JMP AGAIN       [Repeat for next cycle]
```

Delay Program or Subroutine

```
DELAY I  : MVI  B, 20 H      [B ← 20 H]
loop 1   : MVI  C, FF H      [C ← FF H]
loop 2   : MVI  D, FF H      [D ← FF H]
loop 3   : DCR  D
           JNZ  loop3        [continue for 255 times]
           DCR  C
           JNZ  loop 2       [continue for 255 times]
           DCR  B
           JNZ  loop1        [continue for 32 times]
           RET               [Return to instruction after call, when finished]

DELAY II : MVI  B, 10H
           JMP  loop 1       [loop 1 of DELAY I program]
```

- DELAY II is same as DELAY I with loop-1 continue for 16 times where as DELAY-I's loop1 continue for 32 times.
- DELAY I & DELAY II are time delay subroutine used for hold the traffic light for some time.